Thesis for The Degree of Licentiate of Engineering

Coordination and Self-Adaptive Communication Primitives for Low-Power Wireless Networks

Valentin Poirot

Division of Networks and Systems
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2020

Coordination and Self-Adaptive Communication Primitives for Low-Power Wireless Networks

Valentin Poirot

Technical Report No 216L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2020.

"To learn to read is to light a fire; every spelled syllable, a spark."
– Victor Hugo

Abstract

The Internet of Things (IoT) is a recent trend where objects are augmented with computing and communication capabilities, often via low-power wireless radios. The Internet of Things is an enabler for a connected and more sustainable modern society: smart grids are deployed to improve energy production and consumption, wireless monitoring systems allow smart factories to detect faults early and reduce waste, while connected vehicles coordinate on the road to ensure our safety and save fuel. Many recent IoT applications have stringent requirements for their wireless communication substrate: devices must cooperate and coordinate, must perform efficiently under varying and sometimes extreme environments, while strict deadlines must be met. Current distributed coordination algorithms have high overheads and are unfit to meet the requirements of today's wireless applications, while current wireless protocols are often best-effort and lack the guarantees provided by well-studied coordination solutions. Further, many communication primitives available today lack the ability to adapt to dynamic environments, and are often tuned during their design phase to reach a target performance, rather than be continuously updated at runtime to adapt to reality.

In this thesis, we study the problem of efficient and low-latency consensus in the context of low-power wireless networks, where communication is unreliable and nodes can fail, and we investigate the design of a self-adaptive wireless stack, where the communication substrate is able to adapt to changes to its environment. We propose three new communication primitives: *Wireless Paxos* brings fault-tolerant consensus to low-power wireless networking, *STARC* is a middleware for safe vehicular coordination at intersections, while *Dimmer* builds on reinforcement learning to provide adaptivity to low-power wireless networks. We evaluate in-depth each primitive on testbed deployments and we provide an open-source implementation to enable their use and improvement by the community.

Keywords

Internet of Things, IoT, Distributed Systems, Wireless Sensors Networks, WSN, Low-power Wireless Networks, Synchronous Transmissions, Consensus, Reinforcement Learning, DQN

Acknowledgment

Pursuing a PhD is often likened to riding a rollercoaster. You are only aware of the abyssal drop facing you once it is too late to turn, yet the momentum gained along the fall helps you climb back up. Further, unlike preplanned rollercoasters, you are the master of your own track, and can count on the guidance of your collaborators along the way. Thus, I would like to thank my supervisor, Olaf Landsiedel, for his mentoring and methods of leadership, fruitful discussions, as well as his support during the transition to a new country. Most of all, I would like to thank him to offer me the opportunity to work on something I enjoy doing.

I would also like to extend my thanks to the colleagues I met and friends I've made along the way, either at Chalmers or Kiel, during our usual Fika breaks or while I unexpectedly dropped by your office, I hope you managed to catch up on your work after my departure. Thanks (in no particular order) Oliver, Dimitris, Christos, Babis, Romaric, Patrick, Paul-Lou, Georgia, Beshr, Bastian, Thomas, as well as all the others, either back home, scattered across the continent, or beyond the seas, you know who you are. Daily work would not be this smooth without the help of Brigitte, Gerd, Rebecca, and Eva, as well as all the administrative staff involved, and I would like to thank them too.

Finally, I cannot finish before thanking the people that helped me grow into who I am today, my lovely parents and my sister, whom although thousands of kilometers separate me from, always gave me their unconditional support and trust along every step I've chosen. *Merci à tous !*

Valentin Poirot
Kiel, August 2020

List of Publications

Appended publications

This thesis is based on the following publications:

[A] **V. Poirot**, B. Al Nahas, O. Landsiedel
"Paxos Made Wireless: Consensus in the Air"
Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN), 2019, pp. 1-12.
This paper was nominated as a *candidate to the best paper award.*.

[B] P. Rathje, **V. Poirot**, O. Landsiedel
"STARC: Low-power Decentralized Coordination Primitive for Vehicular Ad-hoc Networks"
Third International Workshop on Intelligent Transportation and Connected Vehicles Technologies (ITCVT), part of: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–6.

[C] **V. Poirot**, O. Landsiedel
"Dimmer: Self-Adaptive Network Floods with Reinforcement Learning"
Under submission.

Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] **V. Poirot**, O. Landsiedel
"Poster: Learning to Shine - Optimizing Glossy at Runtime with Reinforcement Learning"
Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN), 2019, pp. 226-227.

Contents

Abstract v

Acknowledgement vii

List of Publications ix

1 Introduction 1
- 1.1 Overview . . . 2
- 1.2 Background . . . 4
  - 1.2.1 Low-power Wireless Networking . . . 4
  - 1.2.2 Synchronous Transmissions . . . 4
  - 1.2.3 Consensus . . . 6
  - 1.2.4 Reinforcement Learning . . . 8
- 1.3 Related Work . . . 10
  - 1.3.1 Guarantees in Low-power Wireless . . . 10
  - 1.3.2 Adaptive Low-power Wireless . . . 12
  - 1.3.3 AI-enabled Wireless . . . 12
- 1.4 Research Problems . . . 14
- 1.5 Contributions . . . 15
- 1.6 Conclusions and Future Work . . . 17

2 Paper A: Paxos Made Wireless: Consensus in the Air 19
- 2.1 Introduction . . . 20
- 2.2 Background . . . 22
  - 2.2.1 Agreement and Consensus . . . 22
  - 2.2.2 The Paxos Basics . . . 22
  - 2.2.3 Multi-Paxos . . . 24
  - 2.2.4 Synchrotron . . . 25
- 2.3 Design Rationale . . . 26
  - 2.3.1 Cost of Paxos . . . 26
  - 2.3.2 Paxos Beyond Unicast? . . . 26
  - 2.3.3 Basic Idea: Wireless Paxos . . . 27
- 2.4 Designing Wireless Paxos . . . 27
  - 2.4.1 Wireless Paxos . . . 27
  - 2.4.2 Wireless Multi-Paxos . . . 29
  - 2.4.3 System Details . . . 30

2.4.4 Design Discussions . . . 31
2.4.5 On the Correctness of Wireless Paxos . . . 31
2.5 Evaluation . . . 32
2.5.1 Evaluation Setup . . . 32
2.5.2 Dissecting Wireless Paxos . . . 34
2.5.3 Paxos and Primitive Latencies . . . 35
2.5.4 Influence of Multiple Proposers . . . 36
2.5.5 Comparing the Cost of Primitives . . . 37
2.5.6 Primitives Consistency . . . 38
2.6 Related Work . . . 39
2.7 Conclusion . . . 40
2.8 Acknowledgments . . . 40

3 Paper B: STARC: Low-power Decentralized Coordination Primitive for Vehicular Ad-hoc Networks 41
3.1 Introduction . . . 42
3.2 Background . . . 43
3.2.1 Related Work . . . 43
3.2.2 Synchronous Communication with Synchrotron . . . 44
3.3 Design . . . 44
3.3.1 Distributed Reservation Coordination . . . 45
3.3.2 Handover Support and Leader Election . . . 46
3.3.3 Platoon Extension . . . 47
3.4 Evaluation . . . 47
3.4.1 Radio Failures . . . 48
3.4.2 Delay Induced by Crossing . . . 49
3.4.3 Traffic Lights Comparison . . . 50
3.5 Conclusion . . . 51

4 Paper C: Dimmer: Self-Adaptive Network Floods with Reinforcement Learning 53
4.1 Introduction . . . 54
4.2 Background . . . 56
4.2.1 Reinforcement Learning . . . 56
4.2.2 Low-power Wireless Bus . . . 58
4.3 An Overview of Dimmer . . . 59
4.4 Designing a RL-based protocol . . . 62
4.4.1 Resource-constrained RL Problem . . . 62
4.4.2 Learning From Traces . . . 64
4.4.3 Distributed Forwarder Selection . . . 65
4.5 System Design: Dimmer, an All-to-All adaptive primitive . . . 66
4.5.1 Architecture . . . 66
4.5.2 Central Adaptivity Control . . . 67
4.5.3 Distributed Forwarder Selection . . . 68
4.6 Evaluation . . . 68
4.6.1 Setup and Methodology . . . 69
4.6.2 Deep-Q Network Features Selection . . . 71
4.6.3 Adaptivity Against Interference . . . 73
4.6.4 Forwarder Selection with MAB . . . 74

4.6.5 Performance on Unknown Deployments 74
4.7 Related Work 76
4.8 Conclusion 77

Bibliography **79**

List of Figures

1.1 Cyber-physical System 2
1.2 Solving consensus with Paxos 7
1.3 The basic Reinforcement Learning model 8

2.1 Executing Paxos 23
2.2 Overview of Synchrotron 25
2.3 Executing Wireless Paxos 28
2.4 Wireless Paxos in action 29
2.5 A snapshot of a typical Wireless Paxos round 34
2.6 Executing Wireless Paxos and Wireless Multi-Paxos in Euratech with 188 nodes 35
2.7 Cost of multiple proposers in Flocklab 36
2.8 Comparing the cost of different primitives 37
2.9 Consensus consistency under injected failure 39

3.1 The STARC middleware 44
3.2 Evaluating STARC at an intersection 48
3.3 Delay decomposition 50

4.1 Adaptive wireless stack 55
4.2 The reinforcement learning model 56
4.3 Adaptivity Policy Control in Dimmer 59
4.4 Forwarder Selection in Dimmer 60
4.5 Glossy as a Markov Decision Process 62
4.6 Dimmer's architecture 66
4.7 Testbed topology 69
4.8 Impact of the number of nodes and historical data as input to the DQN 70
4.9 Tuning the reward function 71
4.10 Adaptiveness to interference 72
4.11 Resilience to interference 73
4.12 Forwarder Selection with Multi-Armed Bandits 75
4.13 Probability to act as forwarder 76
4.14 Porting Dimmer to a new deployment 77

List of Tables

1.1 Conditions for successful synchronous transmissions 4

2.1 Estimating the Cost of Feedback in Euratech with 188 nodes . 31
2.2 Statistics and parameters of testbeds used in the evaluation . . 32
2.3 Slot length of each protocol 32

3.1 Evaluation parameters 49
3.2 Commit success and collisions in the presence of radio failures . 49

4.1 Input vector of Dimmer's DQN 63

1

Introduction

Today, most aspects of our society are interwined with the digital world. From the first interconnection of few computers six decades ago for military and scientific purposes, the Internet has now spread all over the globe. Whether through optic fiber backbones covering the ocean floors, copper wires spanning entire buildings, or via radio waves, information continuously flows around us. This digital world first took over our working places with computers, then our living spaces, before it followed us everywhere through the wide adoption of smartphones.

However, this interconnection between the real, physical world and the Internet did not stop at the palm of our hands. For the last decade, a new trend of connecting even more devices has emerged. By adding sensing and networking capabilities to everyday's objects, it is believed that the *Internet of Things* (IoT) will make our lives more efficient and more sustainable [1]. We can cite, as examples of IoT devices, the current adoption wave of smartwatches, which incorporate heart rate sensors to monitor our health. Connected thermostats and window shutters, which are components of smart homes, aim to reduce our energy consumption in our living places. Soon, connected and autonomous vehicles will free us from the corvee of driving to and from work: cars will communicate together and over the Internet to find the best route, saving us both time and fuel, as well as improving our safety on the road.

The Internet of Things will also play an important part outside of the consumers' houses. Electricity providers can optimize their electrical grid by collecting energy consumption in real-time [2]. Factories can greatly improve efficiency by monitoring their production processes, and reduce waste by detecting faults early [3]. Furthermore, sensors can be deployed in remote places to monitor the wildlife and the environment [4, 5], or up the mountains and volcanoes, collecting precious data that can help us predict avalanches or eruptions [6, 7]. Upon natural disasters, drones can be deployed to help us find survivors quicker [8].

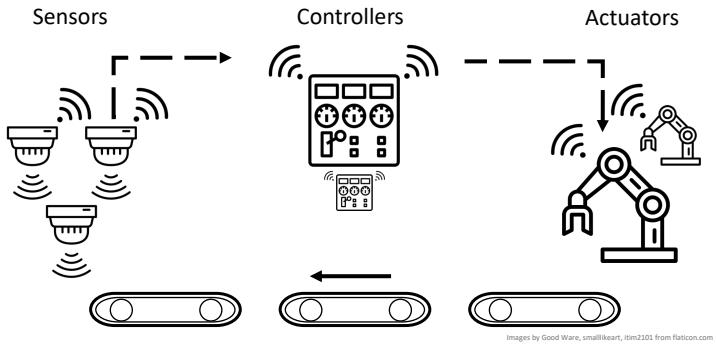


Figure 1.1. Cyber-physical system: monitoring and control can be done via the wireless medium. Sensors send their measurements to controllers, that affect the environment through actuators.

Cyber-physical Systems & Low-power Networking

This pervasive paradigm led to the birth of *cyber-physical systems* [9]. In such structures, the physical and digital worlds not only cohabit but also interact, as depicted in Fig. 1.1. Sensors, installed practically everywhere, periodically collect measurements from physical processes. As example, smart meters deployed in consumer homes locally record electricity consumption. This data produced, sometimes at high velocity, is then securely collected, aggregated, and sent to a central intelligence. Through Big Data analytics or traditional methods, this vast amount of information is processed to obtain a real-time view of the electricity consumption at a city-scale. An electricity provider is then able to adapt energy production to match the needs of its consumers, by issuing commands to production sites, where devices called Actuators act to alter the physical world.

However, in many scenarios, these sensors and actuators can not be wired for power and communication. Sensors deployed across mountain ranges must rely on solar panels and batteries as power sources [7]. Wireless-based sensing systems have been deployed to monitor aging water pipelines when wiring the entire pipeline network was deemed too expensive [10]. When such sensor nodes are not easily accessible for maintenance, the operational lifetime of the network becomes important. Such wireless systems must operate as long as possible, based on their available source of power. Energy consumption must be minimized: low-power, resource-constrained platforms are selected, where the computation power is traded for an energy-efficient hardware. Communication must also be kept to a minimum; the wireless radios are turned off when they are not used [11, 12]. We refer to these deployments as *low-power wireless networks* [13]. In this thesis, we limit the scope of our work to wireless communication in such low-power networks.

1.1 Overview

Context. The Internet of Things is seen as an enabler for a connected and more sustainable modern society. Low-power wireless sensors are deployed in facto-

ries to control production processes [3], in vehicles to improve road utilisation and safety [14, 15], and outdoors to monitor extreme environments [7]. For two decades, the scientific and industrial communities have proposed solutions to provide communication over low-power and lossy multi-hop deployments. Standards have been published (e.g., IEEE 802.15.4 [16], Bluetooth Low Energy [17], LoRa [18]), platforms have been designed (e.g., nRF52 [19], STM32Wx [20]), and competitions have been held to push the field to its limits [21]. Through communication scheduling with Orchestra [22] or synchronous transmissions with Glossy [23], it is possible to obtain high message delivery, in the order of 1 lost packet every 1000 messages, with low radio and energy usage.

Challenges. Many recent applications for the Internet of Things have stringent requirements for their wireless communication substrate: devices must cooperate and coordinate, must perform efficiently under varying and sometimes extreme environments, while strict deadlines must be met. On the road, connected vehicles coordinate to form platoons and cross intersections [15, 24]. In factories, monitoring systems must detect faults early to allow quick reaction [25].

Current distributed coordination algorithms are unfit to solve the requirements of today's wireless applications: solutions borrowed from wired topologies often induce a significant overhead due to mismatched assumptions of the underlying model [26]. While the scientific community has provided a large body of efficient communication primitives for low-power wireless networks [23, 27, 28], few works set to combine their performance with guarantees usually achieved in distributed wired systems and required in mission-critical IoT applications [29, 30].

In addition, the wireless medium is inherently unreliable: external interference can quickly and transiently disturb the medium [31]. If a system is to perform for a decade, its underlying communication primitive must be tuned to ensure performance at a reduced energy cost. Many designers solve this performance-energy tradeoff during the design phase, in a cradle-to-grave fashion, e.g., by selecting Glossy for normal deployments, while favoring Robust Glossy in interfered environments [23, 32]. To ensure energy efficiency in varying environments, a communication primitive must adapt not only to traffic changes, but also to the dynamics of the wireless medium.

Goals. From these starting observations, our goals are two-fold: **(a)** we want to provide complex coordination on top of unreliable low-power wireless networks, including the case of vehicle-to-vehicle coordination, and **(b)** we want to design self-adaptive communication primitives, that are able to react to disturbances to their environment.

Approach. In this thesis, we use experimental computer science methods. We design, implement and evaluate systems, typically network protocols, as a first step towards bringing this vision of pervasive IoT to a state of reality. We make our design and source code freely accessible to enable their use and improvement by the community.

Outline. This thesis is organized into two parts. The first part begins with this high-level introduction to the topic, followed by necessary background notions, a presentation of related work, and a brief summary of the results. The second part of the thesis is a collection of three papers covering the aspects of coordination and adaptiveness in low-power wireless networks.

Table 1.1. Conditions for successful synchronous transmissions in 802.15.4-2.4 GHz.

Effect	Constructive Interf.	Capture Effect
Identical Data?	Identical	Possibly different
Power difference	-	≥ 3 dB
Time difference	≤ 0.5 μs	≤ 160 μs

1.2 Background

In this section, we introduce the core concepts that we build upon. We divide this section into four pillars: **(a)** a brief introduction to low-power wireless networking technologies; **(b)** the concept of synchronous transmissions; **(c)** the problem of consensus in distributed systems, and some well-known solutions; and finally, **(d)** the core concept of reinforcement learning, as well as Q-learning.

1.2.1 Low-power Wireless Networking

With limited energy available to communicate, low-power networking restricts communication to low datarates and limited coverage. Low-power technologies are classified into long- and short-range solutions. Low-power wide area networks (LPWAN), such as LoRa [18] and Sigfox [33], aim to connect battery-powered devices over large areas. They provide ranges up to a few kilometers, but severely limit datarates to tens of kB/s [18]. In contrast, wireless personal area networks (WPAN) provide short-range communication (tens of meters), with higher available datarates (up to few Mb/s). IEEE 802.15.4 [16] and Bluetooth Low Energy (BLE) [17] are two prominent standards for narrowband short-range networking, and both use the 2.4 GHz ISM band, while IEEE 802.15.4 is also able to operate in sub-GHz bands.

Due to their limited energy budget and transmit power, usually around 1 mW, IEEE 802.15.4 devices form mesh topologies to provide multi-hop communication, where some nodes act as relays. Multi-hop communication is usually done with routing, or flooding. In contrast, BLE's main mode is to form single-hop star topologies, with a master and one or more slave platforms. In addition, BLE introduces Bluetooth Mesh, a multi-hop mesh networking mode based on flooding. In the remainder of this thesis, we focus on IEEE 802.15.4 networks operating in the 2.4 GHz band. Moreover, we rely on the flooding principle to communicate over multiple hops.

1.2.2 Synchronous Transmissions

Wireless radios are broadcast-oriented: any antenna in the vicinity receives a radio transmission if the physical channel is good enough. When two transmissions overlap, their physical radio waves add up, often leading to an illegible signal at the receptive end. We refer to this physical behavior as *destructive interference*. For long, the consensus was that overlapping transmissions are destructive, and should be avoided at all costs. CSMA/CA, the access control used in WiFi and Zigbee, relies on this philosophy.

Constructive interference. However, it turned out that not all interference are destructive. If two transmitters transmit the *same data* at the *same time*, the radio waves, which are then identical, superpose in what is called *constructive interference*. Ringwald and Römer showed the first use of such interference in low-power wireless networks with BitMAC, by superposing On-Off Keying (OOK) symbols [34]. Later, A-MAC extended the concept from bits to small packets, by concurrently acknowledging request messages [35]. Glossy goes a step beyond and provides network-wide, fast floods supporting mobile nodes, without the need for expensive routing [23]. Ferrari et al. also showed that, in order to work, Glossy and IEEE 802.15.4-based constructive interference require a synchronisation error smaller than 0.5 μs, which corresponds to a IEEE 802.15.4 chip period.

Further works softened these claims of constructive interference. Wilhehm et al., as well as Liao et al., argue that what we observe is rather non-destructive interference: the received signal can be decoded, but is nonetheless degraded by the collision [36, 37]. The coding scheme used in the physical layer of IEEE 802.15.4, Direct Sequence Spread Spectrum (DSSS), is hypothesised as the reason packets survive such interference.

Other technologies. Al Nahas et al. empirically show that BLE-based synchronous transmissions are also possible, although BLE uses Gaussian Frequency Shift Keying (GFSK) as modulation scheme and does not rely on DSSS [38]. Their results are confirmed by Baddeley et al. In their paper, the authors show that the coded BLE modes offer similar reliabilities as IEEE 802.15.4 in the absence of interference [39]. Lobba et al. demonstrate the feasability of synchronous transmissions using IEEE 802.15.4 ultra-wideband (UWB) radios [40].

Capture effect. Constructive interference is not the only physical behavior in IEEE 802.15.4 we can use to successfully communicate when multiple transmissions overlap. In the case of two overlapping transmissions, if a signal is much stronger than the second at the receiver, the strongest signal can be decoded with high probability. We then speak of the *capture effect*. This effect has been observed long ago for FM transceivers [41]. Dutta et al. found that, for IEEE 802.15.4, a power difference of +3 dB is sufficient to correctly decode a packet consistently [35]. Chaos uses the capture effect to relax the identical data and tight synchronisation requirements of Glossy [42]. By concurrently transmitting *different data* and relying on the capture effect, Chaos provides all-to-all aggregation with low-latency. Codecast and Mixer go beyond and add network coding to provide concurrent many-to-many communication [43, 44].

Table 1.1 summarizes the necessary conditions to obtain constructive interference and utilize the capture effect. In the remainder of this thesis, we use the term Concurrent Transmissions when the sole capture effect is used (i.e., when we transmit different data). Synchronous Transmissions is used for situations where both constructive interference and capture effect are at play (i.e., we transmit the same data). We refer the interested reader to an extensive, recent survey on synchronous transmissions by Zimmerling et al. [45].

1.2.3 Consensus

The field of distributed computing focuses on systems in which participants, distributed across a network, interact to achieve a common goal. One of the most fundamental problems of distributed systems is known as *consensus* [46]. To have a consensus, a group of participants must reach an agreement on a single data value. Coordination of drones, distributed databases, leader election, and state-machine replication are few examples of applications where consensus is required. In fact, many problems in distributed systems can be reformulated as consensus problems, and thus solved if we solve consensus [47]. A correct solution to the consensus problem must have the following properties:

- *Validity:* the agreed value has been initially proposed;
- *Agreement:* all correct processes agree on the same, unique value;
- *Termination:* every process decides in a bounded time; and
- *Integrity:* if all correct processes choose value v, then any correct process must choose v.

FLP and CAP. One of the most influential results in distributed systems is known as the FLP impossibility [46]. Fischer et al. prove that in an asynchronous system, no solution can provide consensus with all the above properties. In few words, their paper shows that if a message can be infinitely delayed, e.g., because of retransmissions, then one participant will never be able to communicate, although it is a correct process. Consensus, as described above, is thus impossible in a fully asynchronous system.

Another impossibility result is known as the CAP theorem [48]. While the CAP theorem is not targeting consensus, but rather read-write storages, its result is nonetheless important to keep in mind during the design of distributed systems. Brewer, the author of the CAP theorem, states that, in a shared-data system, no more than two out of these three properties can be satisfied at the same time: **A**vailability: the system will eventually respond to a request; **C**onsistency: the response is consistent with the latest state; and **P**artition tolerance: the system sustains losses of communication.

Distributed commit. In a distributed system, the commit problem refers to the coordination of a group of participants to agree or reject a given transaction. A transaction is committed if and only if all participants agree to accept it. 2-Phase Commit (2PC) and 3-Phase Commit (3PC) are two notable protocols for distributed commit [49, 50]. Both protocols handle failures in drastically opposite approaches: 2PC is blocking, meaning that no further transactions are accepted unless the current is finalized, while 3PC relies on timeouts to avoid blocking. However, in certain cases, 3PC might lead to inconsistencies in the system, where some participants commit, while other abort the transaction.

Fault-tolerant consensus. Commit is a specialized type of consensus: in the consensus problem, multiple values are initially proposed, and the participants must agree on a single value. Paxos is a solution for fault-tolerant consensus: as long as a majority of nodes are running and are reachable, Paxos eventually achieves consensus [51, 52]. Paxos assumes an asynchronous, non-Byzantine system with crash-recovery: messages can be dropped and delayed, but not tampered with; the network can be partitioned; nodes can crash and

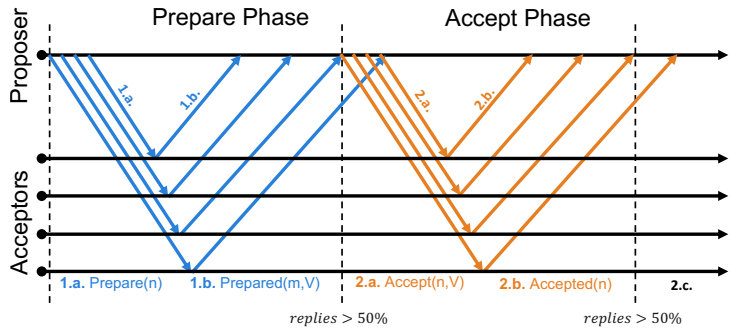


Figure 1.2. Solving consensus with Paxos. A proposer sends a Prepare request with proposal number n. Acceptors reply with the last accepted proposal m with value V. After a majority of Prepared(), the proposer adopts V and sends an Accept request. Proposal n is then accepted by the acceptors.

recover, and have access to persistent storage. However, Paxos requires eventual synchrony to terminate: eventually, a majority of nodes will have bounded message delays.

Processes are classified as follows: **(a)** proposers propose a value to agree on, and act as coordinators for the protocol's execution. Unlike 2PC and 3PC, where at most one coordinator must be present, Paxos supports the presence of multiple proposers. **(b)** acceptors reply to proposers requests by accepting proposals. They informally act as the system's distributed memory. **(c)** learners do not participate in the consensus: they only learn the agreed value once a consensus is met.

The protocol consists of two phases: the *Prepare* phase and the *Accept* phase. The protocol, depicted in Fig. 1.2, is executed as follows:

1. **Prepare Phase**

a. A proposer starts a consensus by generating a unique proposal number n. The proposer broadcasts a *Prepare(n)* request to the acceptors.

b. Upon reception of *Prepare(n)*, an acceptor saves the highest proposal number received so far (*mProposal*). The acceptor replies with the last proposal accepted, if any, and the corresponding accepted value.

2. **Accept Phase**

a. Once hearing from a majority of acceptors, the proposer adopts the value with the highest proposal received, if any. Thus, Paxos ensures that at most one value can be chosen. The proposer broadcasts an *Accept(n, V)* request to all acceptors.

b. Upon reception of *Accept(n, V)* and if $n \geq mProposal$, an acceptor accepts the proposal and associated value, and saves n as new $mProposal$. The acceptor replies with the highest proposal received.

c. If a response with $mProposal > n$ is received, the proposal is rejected. If a majority of replies are received and the proposal is not rejected, the value is chosen.

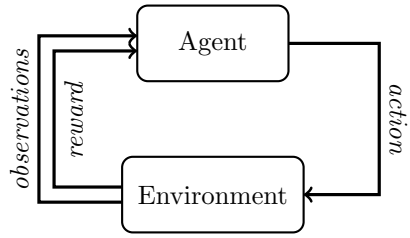


Figure 1.3. The basic Reinforcement Learning (RL) model.

The proposer can therefore inform the learners that a consensus is reached. Using *mProposal* ensures that only the most recent proposal can be accepted, and Step 1.b. ensures that at most one value can be chosen.

In this thesis, we investigate both fault-tolerant consensus and distributed commits in low-power wireless networks.

1.2.4 Reinforcement Learning

The recent advances in artificial intelligence and machine learning have helped reshape the landscape of many scientific and technical fields. First in computer vision, followed by natural language processing and speech recognition, deep learning has been shown to outperform many traditional methods formerly considered as state-of-the-art [53].

Types of learning. Learning techniques can be divided into three broad families, where some algorithms can be classified in more than one (e.g., semi-supervised learning) [53]:

- *Supervised learning*: a dataset of features and labels is available, the goal is to find a generalized mapping function;
- *Unsupervised learning*: an unlabeled dataset is available, the goal is to find if a hidden structure exists within; and
- *Reinforcement learning*: an environment, simulator, or traces are available, the goal is to find a sequence of interactions (a policy) leading to a desired, final state.

In this thesis, we focus on reinforcement learning.

Reinforcement learning. Reinforcement Learning (RL) involves problems where learning is done through interaction [54]. The goal of an RL solution is to learn what to do, i.e., select the best action, in the current situation. Unlike supervised learning, the correct action is unknown both to the system and to the designer: the input data is often unlabeled. However, a RL solver is guided by a *reward* signal. This numerical value quantifies how beneficial the new situation is. The reward signal is used to reinforce the belief of the system whether the chosen action was beneficial or detrimental to the system progress. In some problems, the reward is not immediate. Several consecutive actions are needed to obtain a positive reinforcement. Thus, the goal of an RL solver is to maximize the reward function over time. Sutton and Barto summarize the three main characteristics of RL problems as follows: **(a)** the

problem is a closed-loop system, **(b)** there is a lack of prior knowledge, and **(c)** the consequences of actions play out over extended time periods [54].

Agent and environment. Fig. 1.3 illustrates the basic interaction in RL problems. We refer to the learning method as the *agent*. The agent is surrounded and interacts with its *environment*. The agent can observe the current state of the environment: these observations represent the input features of the system. The agent computes the best action and act. The environment is affected by the action, and new observations and a reward reflect its new state.

Exploration and exploitation. The main challenge faced in RL is the exploration-exploitation trade-off. The goal of an agent is to maximize its reward signal. To do so, the agent chooses actions tried in the past and known to be rewarding. However, to discover such beneficial actions, the agent must also try actions that were never tested before. By *exploring* the environment and trying random actions, the agent can accumulate experiences and build an internal model of how its actions affect the environment. By *exploiting* its accumulated knowledge, the agent can construct a sequence of actions that maximizes the reward.

Markov Decision Processes. Most RL problems can be represented as Markov Decision Processes (MDPs) [55]. MDPs extend Markov chains: in Markov chains, the transition from a state to a new state is represented as a probabilistic distribution. In contrast, in an MDP, the transition is affected both by a decision and by randomness. Formally, a MDP is represented by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ the set of possible actions, $\mathcal{P}$ the transition probability function and $\mathcal{R}$ a reward function. The MDP is said to be finite if $\mathcal{S}$ is finite. Most RL results suppose a finite MDP.

Q-Learning. Finding the best sequence of actions requires an agent to estimate future rewards. The agent seeks to maximize the cumulative reward $R_t \triangleq \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$, where r_τ is the reward obtained when transitioning at time τ, and $\gamma \in [0, 1)$ a constant called the discount factor. A small discount factor will force the agent to maximize short-term, immediate rewards, while a high discount factor will allow the agent to maximize long-term expected rewards.

Q-learning is one of the most popular ways to solve RL problems [54]. In Q-learning, an agent learns an action-value function $Q(s, a)$. This function represents the expected cumulative reward the agent should get when starting in state s, using action a such as:

$$Q(s, a) \triangleq \mathbb{E}\big[R_t \mid s_t = s, a_t = a\big] \tag{1.1}$$

In simple terms, the Q-function evaluates how valuable it is to choose action a in state s in terms of expected reward. If the environment can be modeled as an MDP, then we can find an optimal function $Q^*(s, a)$ that follows Bellman's principle of optimality [56]:

$$Q^*(s, a) = \mathbb{E}\big[r_t + \gamma \max_{a'} Q^*(s', a') \mid s_t = s, a_t = a\big] \tag{1.2}$$

where r_t is the immediate reward received, γ the discount factor presented above, and s' the state achieved after the state s. By iteratively trying actions and receiving rewards, we can update a Q-function that ultimately converges to the optimal $Q^*(s, a)$.

Deep Q-learning. While Q-learning algorithms have historically used a tabular approach in estimating the Q-function [54], deep neural networks have been recently proposed as Q-function approximators [57]. Deep Q-networks (DQN) have been successfully used to solve various problems: Datacenter cooling [58], wireless modulation [59], CSMA/CA optimization [60], etc. The advantage of DQN over tabular approaches is the ability to solve problems with continuous states and the generalization property of neural networks. While RL is not limited to Q-learning and DQNs, we focus on DQNs in the second part of this thesis.

1.3 Related Work

In this section, we present a curated selection of works that represent the current state of the literature in the field of this thesis. We categorize related work in the following aspects: **(a)** providing guarantees in wireless networking, **(b)** adapting communication to external factors, and **(c)** the use of AI and reinforcement-learning methods applied to wireless communication.

1.3.1 Guarantees in Low-power Wireless

Wireless systems are unreliable by nature; external factors such as interference degrade communication, while battery-powered devices are prone to failures. Although a substantial body of work came to term with the situation, designing best-effort solutions, other works focused on providing guarantees to wireless communication.

Failure detection. Node failure can degrade performance. This is particularly true for routing-based solutions, where an unresponsive node can affect the packet delivery of an entire subset of a network. However, communication faults and node failures are sometimes hard to differentiate [61]. Ruiz et al. propose MANNA, a self-diagonistic and self-healing solution to fault management using active requests [62]. In contrast, Miao et al. use correlation patterns to detect possible silent faults in wireless sensor networks [63]. Jhumka and Mottola combine theoretical and systems approaches to tackle the problem of neighborhood view consistency, i.e., the accuracy and correctness of localized neighbor information [30, 64]. While they prove that it is impossible to design a localized solution to strong neighborhood view consistency, they propose a localized algorithm for weak view consistency. By aggregating the 2-hop neighbor information, their solution is able to raise a signal whenever a transient fault or node crash has occured in the vicinity.

Reliable delivery. Even when assuming that nodes do not fail, message delivery is not guaranteed in wireless systems. To tackle this problem, reliable transport protocols, some sharing similarities with TCP, have been proposed. To provide reliable data acquisition for a structural monitoring deployment, Xu et al. propose Wisden, a data transport protocol combining end-to-end as well as hop-by-hop recovery mechanisms [65]. Paek et al. go further with RCRT, and incorporate congestion control in addition to reliable delivery in low-power wireless networks [66]. Kumar et al. endowed TCP with several optimization to fit it to resource-constrained wireless nodes [67]. Building on synchronous transmissions and LWB [27], VIRTUS brings virtual synchrony to low-power

wireless networks [29]. Virtual synchrony provides reliable atomic multicast, i.e., VIRTUS is able to guarantee the message delivery to all recipients of the multicast, while guaranteeing that messages are delivered in order.

Delay-bounded delivery. Time-critical cyber physical systems, such as industrial control systems, have stringent requirements in terms of latency, but often can survive few message losses [68]. Thus, reliable data delivery is not imperative in such deployments. Instead, bounded-delivery protocols fulfill the requested time requirements. Chipara et al. propose a centralized deadline-based scheduler to ensure on-time delivery [69]. Li et al. extend it for emergency alarms over wireless [25]. With TTW, Jacob et al. show that wireless solutions can replace wired field buses in industrial settings [68]. By building on synchronous transmissions, TTW provides end-to-time timing predictability, high reliability and low-latency.

Coordination. Agreement has also been studied in opportunistic and ad-hoc networks [70, 71]. Turquois provide consensus in the presence of Byzantine faults in single-hop networks [72]. Köpke investigates the performance 2-Phase Commit (2PC) for WSNs [26]. The author shows that both low-latency and high-reliability MAC and routing layers are necessary to provide commit semantics with sufficient performance in sensor networks. With RedMAC and the routing protocol Net, Köpke provides a wireless stack able to run the original 2PC for 16 participants under 2 min. However, the author only hints at additional improvements, such as multicast and aggregation, but does not provide a dedicated 2PC design tailored to wireless communication. In contrast, Borran et al. extends Paxos with a new communication layer for 802.11 opportunistic networks [73]. The authors build a tree to route and collect acceptor responses. With JAG, Boano et al. take full advantage of the destructive behavior of jamming signals, as JAG uses jamming to acknowledge one-hop agreement requests [74].

Building on top of synchronous transmissions and reusing concepts from Chaos [42], Agreement in the Air (A^2) introduces 2-Phase Commit (2PC) and 3-Phase Commit (3PC) to low-power lossy networks, providing commit guarantees with low-latency and high reliability [75]. Later, Spina et al. propose a new approach to 2&3PC named XPC [76], this time combining Chaos with Glossy floods. Compared to A^2, their approach has the advantage of terminating faster during aborts, while providing similar latencies for successful commits. Independently of Paper A of this thesis (see Chapter 2), Spina worked on a Paxos implementation, reusing the approach used in XPC, called WISP [77]. The author obtained performance of the same order as the one provided in our Paper A for Paxos, but higher latency variation in their Multi-Paxos implementation.

Vehicle coordination. Vehicle-to-vehicle communication usually relies on cellular technologies (e.g., LTE or 5G) or IEEE 802.11p to communicate [78]. In contrast, in Chapter 3, we argue that using low-power radios allow more participants (e.g., bikes, pedestrians) to coordinate. In the case of a road intersection, a coordination protocol must ensure the safety of its participants, while minimizing the delay a vehicle (or pedestrian) must observe before being able to cross. Dresner and Stone propose AIM, a centralized intersection management protocol relying on the cellular infrastructure [79]. Ferreira et al. propose Virtual Traffic Lights, a decentralized solution where cars elect a

leader mimicking traffic lights to control the intersection [15]. In contrast, we propose in Chapter 3 a leader-based solution using commit semantics, where access to the intersection is based on the waiting time of the participating vehicles.

1.3.2 Adaptive Low-power Wireless

In this thesis, we use the term adaptivity to refer to the ability of a system to adapt to external changes in the environment. However, adaptivity has also been used in the literature as the ability for a solution to adapt to changing data traffic. In this section, we present protocols in low-power wireless that adapt to either changes in traffic or their environment.

Traffic-oriented. LWB is the de-facto protocol for many-to-many low-power wireless communication [27]. LWB works in communication rounds, and embeds a central scheduler. Based on the requested bandwidth, the scheduler adapts its round interval and slot attribution, thus minimizing energy consumption while fulfilling required traffic demands. Blink builds on top of LWB and goes a step beyond, providing delivery guarantees to deadline-based flows while maintaining adaptivity to changing traffic demands [80]. Because LWB, and thus Blink, are based on network-wide Glossy floods [23], they are impervious to node mobility and, to some extent, link quality. However, we show in our Paper C (see Chapter 4), that interference nonetheless degrades the performance of LWB.

Environment-oriented. Collision avoidance through channel assessment is a common approach to adapt to the medium in wireless communication and is standard in IEEE 802.15.4 [16], although channel assessment has limited capabilities. Several works propose new metrics to measure and quantify link quality, and its evolution through time. Srinivasan et al. as well as Munir et al. propose to use link burstiness as routing metric [31,81]. Noda et al. propose the Channel Quality metric as a measure of channel availability over time [82]. MUSTER provides time-adaptive routing for data collection where multi-sink are present [83]. In MUSTER, the available energy is taken into account in the routing process in addition to the link quality.

1.3.3 AI-enabled Wireless

For long, heavy computation processes such as Big Data analytics and machine learning training and inference have been offloaded to powerful servers hosted in the Cloud. However, a recent trend has pushed back computation closer to the data source, in a bid to reduce network congestion and improve application performance [84]. As a result, a body of work investigates the performance of machine learning inference, sometimes even training, from the network edge up to the sensor nodes themselves. In this section, we focus on the different techniques used to embed neural networks onto resources-scarce platforms, as well as the current deployment of reinforcement learning methods for wireless communication.

Embedding neural networks. Neural networks are notorious for being compute-intensive as well as memory-intensive. Deep CNNs for image recognition often feature several million parameters. With their SparseSep framework,

Bhattacharya and Lane optimize a sound recognition deep network containing 1.8 M parameters on a Cortex-M0 constrained platform featuring 8 kB of memory only [85]. The authors use sparse factorization of weights between layer, formulating the factorization as a sparse dictionary learning problem. Additionally, the use of separable filters allow CNNs to also be executable on constrained devices. DeepX, a software accelerator for inference on mobile devices, uses singular value decomposition to reduce weight representation in memory [86]. In addition, DeepX is able to segment the model into subparts, that are distributed to the several compute centers (e.g., CPU, GPU, LPU...) available on the platform, thus greatly improving inference time. The same year, Han et al. combine sparse representation with pruning to save further resources [87]. First, the authors prune weights with negligible impact on performance, using a threshold, thus obtaining a sparse representation of the original weights. Then, the remaining parameters are quantized into a lighter representation. Finally, a Huffman coding for lossless compression is applied on top of the two previous steps to minimize further the memory footprint of the deep neural architecture.

In contrast, DeepIoT uses dropout to minimize the number of elements per layer, thus keeping a dense matrix representation [88]. DeepIoT iteratively searches the optimal per-neuron dropout probability using a so-called compressor neural network. The compressor network, implemented as a recurrent neural network using LSTMs, takes as input the layer weights, and outputs the dropout probability, by searching for redundancy. FastDeepIoT investigates the non-linear relationship between neural architecture and inference time [89]. By evaluating certain architectures on the target device, FastDeepIoT produces a tree of requirements that leads to a shorter inference time, and is able to compress any given neural network to meet these constraints. While all those solutions focus on fitting neural networks to embedded platforms, Bonsai is a tree learner that produces a shallow and sparse non-linear tree with minimal memory impact [90]. Using only 0.5 kB, Bonsai is able to obtain up to 94% accuracy on the MNIST-2 dataset.

RL-enabled wireless. Recently, some works have started to investigate how to apply reinforcement learning to wireless protocol control. Amuru et al. as well as Mastronarde et al. learn contention for 802.11 CSMA/CA, both using post-decision state-based learning [60,91]. At the physical layer, Vrieze et al. set out to entirely learn a modulation scheme [59]. Using a known preamble, policy-gradient methods, and two independent agents, their system iteratively tries to modulate and demodulate the transmitted preamble over a noisy channel, until both agents are able to reconstruct the message. The agents are able to converge to a rotated version of 16-QAM, without any prior knowledge of modulation techniques. Dakdouk et al. propose a channel selection scheme for IEEE 802.15.4-TSCH [92]. Using multi-armed bandits, their model selects the channel where the next transmission is most likely to succeed. Closer to our work, Zhang et al. use multi-armed bandits to optimize Glossy floods [93]. in their work, each IoT platform runs Exp3 independently, where an arm represent the number of retransmission in a flood. However, they assume that the wireless medium does not suffer any sudden changes, while we assume sudden interference in our Chapter 4.

Like us, Restuccia and Melodia argue that deep-RL can be used to recon-

figure the wireless stack to adapt to changes to the wireless medium [94]. They propose DeepWiERL, a hardware-software framework to execute and train DRL on IoT platforms. By using FPGAs, they are able to execute inference in a timely manner, and use transfer learning to bootstrap training on new tasks. In contrast, we do not rely of additional hardware such as FPGAs in our work, but share the vision of using RL to reconfigure the network stack at runtime. Joseph et al. go a step further and argue for self-driving radios, a paradigm where the wireless stack learns its optimal configuration from high-level only specifications of the scenario [95]. However, they simply create and train a new DQN whenever a new scenario is defined. However, using a DQN per scenario is memory-intensive, and requires a training environment and a specification of each scenario. In contrast, our self-adaptive network stack updates its internal parameters to adapt to a changing environment.

1.4 Research Problems

The Internet of Things is a paradigm in which objects are getting augmented with sensing and networking capabilities. Within the IoT ecosystem, low-power wireless networks stand out for their use of resource-constrained hardware, their strong energy limitations and their reliance on low-power, short-range unreliable wireless communication. Many applications foreseen for such networks, such as drone control, wireless closed-loop industrial systems, and safety-critical monitoring infrastructures, require complex coordination between devices.

Solutions borrowed from distributed wired settings are unfit to solve wireless coordination and cause a high overhead due to mismatched assumptions on the underlying communication model. In contrast, many energy-efficient primitives developed for low-power wireless networks lack the guarantees required by mission-critical applications. Consequently, this thesis first focuses on answering the following question:

RQ1: *How can we provide safe, low-latency consensus primitives for low-power wireless networks where connectivity is unreliable and nodes can fail?*

We provide our answer to this question in Papers A and B. Note that Paper A provides an approach to fault-tolerant consensus, while Paper B focuses on safe vehicle-to-vehicle coordination at intersections.

Once the case of complex coordination is settled, we decide to take a broader look at the landscape of communication primitives for low-power wireless networks. We notice that many solutions are static by design, and can be categorized into two classes: either **(a)** protocols are optimized for the normal case, and break once harmful interference comes in, or **(b)** protocols are over-provisioned for the worst-case scenario, thus wasting precious resources the rest of the time.

In the remainder of this thesis, we set out to answer this second question:

RQ2: *How can we design efficient communication primitives able to react and adapt to changes to the wireless medium?*

Paper C gives our first insights regarding adaptiveness in low-power wireless networks.

1.5 Contributions

We summarize in this section the papers that constitute the second part of this thesis.

Paper A - Paxos Made Wireless: Consensus in the Air

In this paper, we address the problem of fault-tolerant consensus in low-power wireless networks. While consensus is a mature field of research in traditional, wired networks, we argue that the solutions proposed there do not satisfy the requirements of resource-constrained wireless networking.

We introduce *Wireless Paxos*, a new flavor of Paxos fitted to the characteristics of low-power wireless networking: we show that Paxos can be transformed from a unicast scheme to a many-to-many scheme, which can be efficiently executed in low-power wireless networks. We co-design the consensus algorithm along with the lower layers of the network stack to greatly improve the latency of consensus and have a tighter control on the transmission policy. The overall result is a broadcast-driven consensus primitive using in-network processing to compute intermediate results in Paxos. Our solution builds on top of Synchrotron [75], a kernel for concurrent transmissions inspired by Chaos [42], providing a basis for highly reliable and low-latency networking in low-power wireless with support for in-network processing.

Our results show that Wireless Paxos requires only 289 ms to complete a consensus between 188 nodes in testbed experiments. Furthermore, we show that Wireless Paxos stays consistent even when injecting failures.

Personal contribution. I am the lead designer and implementer of Wireless Paxos and its extension, Wireless Multi-Paxos. Additionally, I am the main designer of its evaluation, and the main author of the paper. The chapter was published as a paper in the Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN), 2019 [96], and its source code is available on GitHub[1]. This paper was nominated as *candidate to the best paper award* at the conference.

Paper B - STARC: Low-power Decentralized Coordination Primitive for Vehicular Ad-hoc Networks

This paper revisits the network-wide consensus with a focus on vehicle-to-vehicle (V2V) communication. V2V communication is expected to improve road usage efficiency through cooperative driving, platooning, and autonomous intersection management. By deploying V2V solutions, it is possible to reduce the infrastructure cost of the road network, e.g., by replacing traffic lights and traffic signs by their digital counterparts. However, such V2V coordination protocols must ensure the safety of all road users, as well as perform timely, and offer the possibility for all road users to participate.

We introduce STARC, a decentralized reservation-based protocol that uses cheap, low-power wireless radios to enable energy-efficient vehicle-to-vehicle communication. We build our coordination protocol on top of Synchrotron,

[1] Available at https://github.com/iot-chalmers/wireless-paxos

a low-latency and energy-efficient communication primitive for all-to-all communication [75]. With STARC, traffic participants reserve lanes to cross the intersection. We provide transaction semantics, and all participants coordinate to commit on a shared access pattern jointly. As a result, vehicles have unique access to different parts, i.e., lanes, of the intersection, ensuring that at most, one car can use a given lane. Once a car has crossed the intersection, it releases its reservation and leaves the communication network, freeing its resources for other vehicles. Our design and implementation for IEEE 802.15.4 radios allows STARC to operate on energy-restricted devices to support all road users, including cyclists and pedestrians.

We show that STARC reduces average waiting times by up to 50% compared to a fixed traffic light schedule in traffic volumes with less than 1000 vehicles per hour. Moreover, we show that the protocol supports dynamic priority strategies and illustrate a platoon extension that allows STARC to outperform traffic lights even with traffic loads of over 1000 vehicles per hour.

Personal contribution. I am a co-designer of STARC, and acted as a supervisor during the implementation and evaluation. Additionally, I participated in the write-up of the paper. The chapter was published as a workshop paper in the Third International Workshop on Intelligent Transportation and Connected Vehicles Technologies (ITCVT), 2020 [97], and its source code is available on GitHub[2].

Paper C - Dimmer: Self-Adaptive Network Floods with Reinforcement Learning

In this paper, we observe that many low-power protocols are invariant to their environment dynamics and deal with interference through over-provisioning. Instead, we argue that low-power wireless networking should adapt to the wireless medium to meet a target performance, even under varying conditions, while still ensuring energy efficiency.

We introduce Dimmer as a self-adaptive, all-to-all communication primitive. We build Dimmer on top of LWB and we use deep Reinforcement Learning to tune the flooding parameters to match the current properties of the medium. By learning how to behave from unlabeled traces, Dimmer adapts to different interference types and patterns, and is even able to tackle previously unseen interference. Through Dimmer, we share insights on how to efficiently design AI-based systems for constrained devices, and evaluate our protocol on two deployments of 18 and 48 resource-constrained sensor nodes (4 MHz CPU, 10 kB RAM), showing it improves reliability under WiFi interference and IEEE 802.15.4 jamming.

We show that Dimmer obtains similar performance as LWB in the absence of interference, while maintaining 80% reliability under 30% interference. Further, we demonstrate the generality of Dimmer by evaluating our DQN on a second testbed, featuring previously unseen WiFi interference.

Personal contribution. I am the lead designer and implementer of Dimmer. Additionally, I am the main designer of its evaluation, and the main author of the paper. The chapter is under submission, and the code will be released after publication.

[2] Available at https://github.com/ds-kiel/starc

1.6 Conclusions and Future Work

Throughout this thesis, we set out to study the problems of coordination and adaptivity in low-power wireless communication. As cyber-physical systems are being deployed across the globe for a variety of new and ever more complex tasks, some critical applications require stronger communication guarantees than the current state of the art provides, while others rely on smart primitives to operate under all conditions, at reduced energy costs. We propose three new communication primitives: Wireless Paxos brings low-latency fault-tolerant consensus to low-power wireless, STARC is a middleware for vehicular intersection management with safety guarantees, while Dimmer is an AI-enabled, adaptive many-to-many communication substrate.

Personal Reflections. With this thesis, we take a step towards providing guarantees to low-power wireless networking. Although Wireless Paxos is a general solution to consensus, we agree that not all coordination applications can be solved with a majority-based approach, as we demonstrate it with STARC. Similarly, recent work published after Wireless Paxos show that other approaches are also worth investigating [77]. Furthermore, synchronous transmissions bring us a step toward the synchronous model studied in distributed systems, as discussed by Zimmerling et al. [45]. Solutions for the asynchronous model, e.g., Paxos, are typically more expensive than their synchronous counterparts. Thus, it is possible to leverage synchronous transmissions to design more efficient coordination primitives.

Adaptivity has played an important role in communication engineering for decades, as the numerous TCP and WiFi rate control approaches demonstrate. In contrast, the low-power networking community prefers to focus on dependability, the ability for communication to survive under strong RF interference, as showcased by the EWSN dependability competitions [21]. We believe that there is a need for adaptive low-power networking: communication should be both energy-efficient, i.e., not waste energy under good conditions, and dependable, i.e., not suffer losses under strong RF interference. With Dimmer, we take the first steps to reconcile these two, often antagonistic, goals. Furthermore, we argue that artificial intelligence, and specifically reinforcement learning, is a possible enabler for adaptive low-power networking. Few works investigate the applicability of RL in the wireless setting. Dimmer is a first milestone towards proving the possible benefits and downfalls of RL in adaptive wireless networking, but many challenges are still ahead.

Future Work. In the future, we set out to investigate further the aspect of adaptivity in low-power wireless networks. Channel diversity has been shown to improve dependability of wireless communication, and we want to investigate the advantages of channel blacklisting and adaptive frequency hopping in improving further energy efficiency. Additionally, autonomous, self-forming, and self-configuring networks is a desirable goal of low-power networking, and is achievable once coordination is possible. Finally, a recent work by Al Nahas et al. empirically demonstrate the feasibility of synchronous transmissions over Bluetooth Low Energy [38]. This opens a new promising research direction, that we believe is worth investigating further.

2

Paxos Made Wireless: Consensus in the Air

V. Poirot, B. Al Nahas, O. Landsiedel

Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN), 2019, pp. 1–12.

PAPER A

Abstract

Many applications in low-power wireless networks require complex coordination between their members. Swarms of robots or sensors and actuators in industrial closed-loop control need to coordinate within short periods of time to execute tasks. Failing to agree on a common decision can cause substantial consequences, like system failures and threats to human life. Such applications require consensus algorithms to enable coordination. While consensus has been studied for wired networks decades ago, with, for example, Paxos and Raft, it remains an open problem in multi-hop low-power wireless networks due to the limited resources available and the high cost of established solutions.

This paper presents *Wireless Paxos*, a fault-tolerant, network-wide consensus primitive for low-power wireless networks. It is a new flavor of Paxos, the most-used consensus protocol today, and is specifically designed to tackle the challenges of low-power wireless networks. By building on top of concurrent transmissions, it provides low-latency, high reliability, and guarantees on the consensus. Our results show that Wireless Paxos requires only 289 ms to complete a consensus between 188 nodes in testbed experiments. Furthermore, we show that Wireless Paxos stays consistent even when injecting controlled failures.

2.1 Introduction

Context. Many applications in low-power wireless networks need to reach an agreement among themselves before an action can be performed. Mission critical systems are one typical example of such applications, since conflicting commands can have important and possibly harmful consequences. For instance, a swarm of Unmanned Aerial Vehicles (UAVs) must agree on a common destination [98]; while centrally computed transmission schedules in wireless sensor networks (WSNs) have to be agreed on and distributed by the nodes in the network [99].

However, not all faults can be avoided in wireless networks. Message loss and devices running out of battery are common failures seen in deployments. Thus, an agreement ensures that at most one action is chosen, even if some devices cannot participate. UAVs operating on limited batteries must agree on a common trajectory, even if some UAV ran out of power along the way; and an updated schedule must be agreed upon and used in a WSN, even if some nodes disappeared since the last agreement.

Efficient and highly reliable dissemination protocols have been proposed in the literature [23, 100]. However, they do not provide the same guarantees ensured by an agreement. A Glossy initiator cannot detect network segmentation, for instance, and would be unaware that its command was received by a small subset of the network only. In a swarm of UAVs, it is preferable that as many drones as possible continue on their agreed trajectory, rather than only the few that managed to receive the disseminated command. Reaching a consensus is therefore primordial to ensure the correct and optimal behavior of such applications.

Consensus refers to the process of reaching an agreement. A consensus is achieved once participants agree on a single, common decision, from a set of initial values. Consensus is challenging in the presence of failures (node crashes, message losses, network partitions, etc.). It is even proven that consensus is impossible in a fully asynchronous setting [46], where one node might never be able to communicate.

Many solutions to the consensus problem have been proposed in the literature [51, 101, 102]. Paxos was one of the first protocols to provide consensus [51, 52]. It is (non-Byzantine) fault-tolerant and proven to be correct: Paxos will lead to a correct consensus as long as a majority of nodes are participating. Due to the properties of Paxos, all nodes will eventually learn the correct value as long as a majority accepted the decision. Paxos is often used in an extended and optimized form, Multi-Paxos [51], which allows nodes to agree on a continuous stream of values and enables state machine replication. For example, UAVs can continuously coordinate their next destination with Multi-Paxos.

Today, Paxos and Multi-Paxos have become the default protocols to ensure consistent data replication within data-centers. They are used in many modern deployments, for instance Google's Chubby locking mechanism [103] and their globally distributed database Spanner [104], Microsoft's data-center management Autopilot [105], and IBM's data-store Spinnaker [106].

Challenges. The complexity of Paxos and its many required interactions pose key challenges in low-power wireless networks. Devices in WSNs have strong resource-constraints in terms of bandwidth, energy, and memory. Ra-

dios are, for example, commonly duty-cycled to save energy [11, 12, 107]. In contrast, Paxos requires many message exchanges and a high bandwidth to reach consensus.

In addition, links are highly dynamic and unreliable in low-power wireless communication [31]. Paxos is resilient to these network faults by design, but many of its implementations use end-to-end routing, which induces an overhead to the consensus. Paxos has been initially designed for wired networks, and is therefore heavily influenced by its unicast structure. Later work shows that Paxos can be partially executed with multicast [73], or by introducing an additional logical ring to reduce communications [108]. However, these approaches rely on unicast for parts of the algorithm.

In contrast, low-power wireless networks are broadcast-oriented networks where each transmission can be received by all neighboring nodes. Executing unicast-based schemes in wireless networks usually induces higher costs, especially in multi-hop networks. Moreover, multi-hop networks also provide opportunities for data-aggregation and computation of intermediate results, which are not part of Paxos' design rationale.

Approach. In this paper, we bring fault-tolerant consensus to low-power wireless networks. We propose *Wireless Paxos*, a new flavor of Paxos fitted to the characteristics of low-power wireless networking: we show that Paxos can be transformed from a unicast (or multicast) scheme to a many-to-many scheme, which can be efficiently executed in low-power wireless networks. We co-design the consensus algorithm along with the lower layers of the network stack to greatly improve the latency of consensus and have a tighter control on the transmission policy. The overall result is a broadcast-driven consensus primitive using in-network processing to compute intermediate results in Paxos. Our solution builds on top of Synchrotron [75], a kernel for concurrent transmissions inspired by Chaos [42], providing a basis for highly reliable and low-latency networking in low-power wireless with support for in-network processing.

Contributions. This paper makes the following contributions:

- By distributing parts of the proposer logic, we show that Paxos can be expressed as a many-to-many communication scheme, rather than a partially multicast scheme;
- We present *Wireless Paxos*, a new flavor of Paxos specifically designed to address the challenges of low-power wireless networks, and *Wireless Multi-Paxos*, an optimized extension of Wireless Paxos for continuous streams of agreed values for constrained devices;
- Both primitives are ready to use by any application as an open-source library on GitHub[1];
- We implement and evaluate our contributions on two testbeds, composed of 27 and 188 nodes, and compare our results to solutions from the literature.

The remainder of the paper is organized as follows. Sec. 2.2 introduces consensus, Paxos, and concurrent transmissions. Sec. 2.3 gives an overview of our design. Next, Sec. 2.4 dives into Wireless Paxos and Sec. 2.5 evaluates our contributions. We discuss related work in Sec. 2.6 and conclude in Sec. 2.7.

[1] Available at: https://www.github.com/iot-chalmers/wireless-paxos

2.2 Background

This section introduces the necessary background on consensus and concurrent transmissions. We begin with an overview of consensus. Then, we present Paxos and Multi-Paxos. Finally, we introduce concurrent transmissions and Synchrotron.

2.2.1 Agreement and Consensus

In distributed systems, a consensus refers to the problem of reaching agreement among a set of participants. A consensus is complete if, in the end, nodes agree on the same decision. To be correct, a consensus algorithm must fulfill certain properties: the final value must be valid, i.e., it was proposed at the beginning of the algorithm (Validity); each node must eventually make a decision (Termination); and at most one value can be agreed upon, i.e., the result of the agreement must be consistent among the participants (Agreement) [109].

Dissemination is no consensus. Dissemination allows a node to share a value with the entire network, often (but not always) in a best-effort manner. As such, dissemination does not – *and is not meant to* – solve consensus. For example, Glossy [23] provides high reliability, and a unique value is present if at most one flood initiator is in the system. However, even if a node missing a value is aware of the failed flood, it will never be able to recover the correct command.

2 and 3-Phase Commit. Common algorithms for agreement (but not consensus) are 2-Phase Commit (2PC) and 3-Phase Commit (3PC) [49, 50]. Both algorithms are used to solve the problem of commit, i.e., whether a transaction should be executed by all nodes, or none. 2PC works by first requesting and collecting votes from all nodes in the network, and then disseminating the result of the decision; while 3PC adds an intermediary phase to dissociate the decision from the commit. Both protocols handle failures by aborting or blocking, i.e., by delaying the decision to maintain consistency. However, 3PC might lead to inconsistencies.

Fault-tolerant consensus. Paxos provides consensus. While many values can be initially proposed, it ensures that at most one value is chosen. Eventually, all nodes will learn the decision if progress is not impeded. To do so, Paxos relies on a majority of responses to make the decision, rather than from the entire network, thus handling (non-Byzantine) failures (e.g., message losses, network segmentation, node crashes).

2.2.2 The Paxos Basics

Paxos is a fault-tolerant protocol for consensus. It assumes an asynchronous, non-Byzantine system with crash-recovery, i.e., it handles both process crash and recovery (a persistent storage is needed), but not misbehaving nodes or transient faults; delayed or dropped messages, but not corrupted messages; and network segmentation. The protocol guarantees that, if a majority of nodes runs for long enough without failures, all running processes will agree on the same proposed value. For example, the value can be the destination point for UAVs, or the network configuration in WSNs.

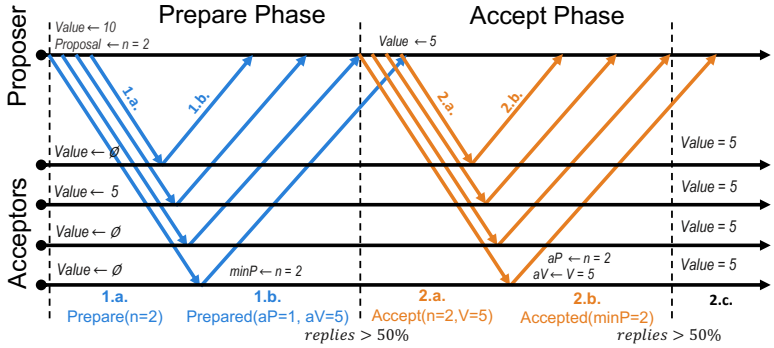


Figure 2.1. Executing Paxos: a proposer wants to propose the value V=10 to four acceptors. It sends a Prepare request to all acceptors *(1.a)* with proposal number n=2. An acceptor replies with a Prepared message *(1.b)* with the most recently accepted proposal, here proposal aP=1 with value aV=5. Once a majority of replies are received by the proposer, it adopts the highest value received, and sends an Accept request *(2.a)*. Upon reception, an acceptor accepts the value *(2.b)* and replies with an Accepted message and the highest proposal received so far. After a majority of Accept messages, the value is chosen.

Roles. A node can act as three different roles: *Proposer*, *Acceptor*, and *Learner*. Nodes can implement more than one role. Proposers propose a value to agree on and act as coordinators for the protocol's execution. Acceptors, unlike in 2PC, don't "vote", but act, in a very informal manner, as the system's "fault-tolerant memory": they reply to proposers requests by accepting proposals. Learners do not participate in the consensus: they only learn which value has been chosen by the acceptors once a consensus is met. Unlike 2PC and 3PC, where at most one coordinator must be present, Paxos can tolerate the presence of multiple proposers, at the cost of impeding the progress of the agreement.

Execution. The protocol consists of two phases: the *Prepare* phase and the *Accept* phase, as depicted in Fig. 2.1. The protocol is executed as follows:

1. **Prepare Phase**

a. Any proposer starts the protocol by choosing a value V to agree upon and a unique proposal number n (i.e., no other proposer can choose the same proposal number n). A *Prepare(n)* request is sent to every acceptor.

b. Upon reception of a *Prepare(n)* request, an acceptor will save the proposal number n if and only if it has never heard any higher proposal number *minProposal* before; i.e., if $n > minProposal$, then $minProposal \leftarrow n$. The acceptor only replies to the request if the precedent condition is met, meaning that the node is promising not to reply to any request with a lower proposal number anymore. The acceptor returns both the last proposal *accepted* by that process, noted *acceptedProposal*, if any has been accepted so far, and the corresponding value, noted *acceptedValue*.

2. **Accept Phase**

a. Upon hearing from a majority of acceptors, the proposer *adopts* the value with the highest proposal number, such as $V \leftarrow acceptedValue$, if any has been received. This condition ensures that at most one value can be chosen by the system. The proposer switches to the Accept phase and sends an *Accept(n,V)* request to all acceptors.

b. Upon receiving an *Accept(n,V)*, an acceptor *accepts* the value V if and only if the proposal number n is higher or equal to the proposal number the process has prepared for, namely *minProposal*. If the condition is true, the acceptor saves the proposal number n as its highest proposal number heard and as its accepted proposal, and the value V as its accepted value, i.e., if $n \geq minProposal$, then $minProposal \leftarrow n$, $acceptedProposal \leftarrow n$ and $acceptedValue \leftarrow V$. Regardless of the result of the condition, the acceptor replies to the request with the highest proposal heard (*minProposal*).

c. Upon receiving at least one reply with $minProposal > n$, the proposer knows that its value has been *rejected*. This also means at least one other proposer is present, and the process can either restart the protocol with a higher proposal number n to compete or let the other proposer win. If the proposer received a majority of replies and no rejection, the value is *chosen*.

The proposer can therefore inform the learners that a value has been chosen by the consensus algorithm. Using *minProposal* ensures that only the most recent proposal can be accepted and the data returned at step 1.b. ensures that at most one value can be chosen.

2.2.3 Multi-Paxos

Using the protocol described above leads to the agreement and dissemination of a single value. Due to the properties of the protocol, any additional execution will lead to the same value being adopted. Being able to agree on a sequence of values is a desirable property that Paxos, in its simple form, cannot satisfy. We can, however, run several *rounds* of Paxos to achieve this result. A multi-round version of Paxos is called *Multi-Paxos*. More importantly, Multi-Paxos allows state machine replication, i.e., replicating operations across processes such as all replicas are identical. For example, the stream of values can represent intermediary waypoints towards the final destination for UAVs, or the evolution of the network configuration over time for WSNs. For details on Multi-Paxos, we refer the reader to the original paper [51].

Executing multiple rounds. A Paxos execution is now identified by its *round number*. Many rounds can be executed simultaneously or sequentially. In the former case, messages of multiple rounds can be merged into one unique message in order to save network resources.

Using a unique proposer. In many applications, nodes are stable enough to keep a unique proposer for a relatively long period of time. In doing so, the Paxos execution can be simplified to its sole Accept phase, the Prepare phase being used only to prepare acceptors to listen to that specific proposer. The Prepare phase is executed at the beginning of Multi-Paxos or after a crash of the proposer, and successive rounds execute only the Accept phase for value

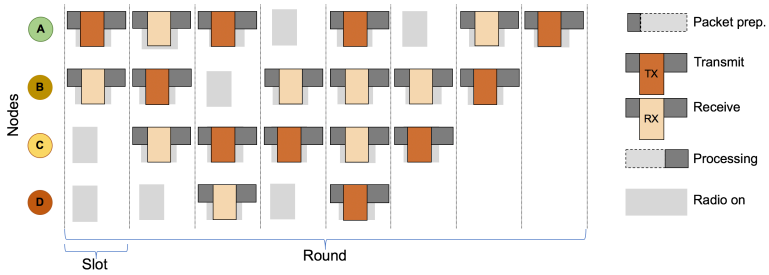


Figure 2.2. Synchrotron Overview: Synchrotron schedules flooding rounds for network-wide dissemination, collection or aggregation, as requested by the application. A round is composed of consecutive slots, during which a node can either transmit, receive, or sleep, and processes the data following a per-application logic.

adoption. The protocol does not break in the presence of multiple proposers since acceptors will only accept the highest proposal and inform lower proposals of their rejection.

Executing Multi-Paxos. Fig. 2.1 shows the execution of Paxos. With Multi-Paxos, the Prepare phase (blue) is executed once at the beginning, and is followed by many Accept phases (orange), until the proposer fails.

2.2.4 Synchrotron

Due to the broadcast nature of wireless communications, concurrent transmissions of nearby nodes inherently interfere with each other. However, when such transmissions are precisely timed, one of them can be received, nonetheless. We briefly introduce the concept of capture effect, and present Synchrotron, the communication primitive used in this work.

Capture effect. Nodes overhearing concurrent transmissions of different data can receive the strongest signal under certain conditions; this is known as the capture effect [41]. To achieve capture at the receiver with IEEE 802.15.4 radios, the strongest signal must arrive no later than 160 μs after the first signal and be 3 dBm stronger than the sum of all other signals [35].

Synchrotron. Agreement in the Air (A^2) [75] extends the concepts of Chaos, a primitive for all-to-all aggregation [42], to network-wide agreement. A^2 introduces Synchrotron, a new transmission kernel using high-precision synchronization. Like Chaos, Synchrotron operates periodically within rounds, which we refer to as "flooding" rounds to distinguish from Paxos rounds. A (flooding) round is composed of slots, in which nodes concurrently transmit different data, while the reception relies on the capture effect. In-network processing is applied after a successful reception and the result of the computation is then transmitted during the following slot, until all progress flags are set, denoting the participation of the different nodes. Fig. 2.2 presents an overview of Synchrotron's inner working.

Our solution reuses the Synchrotron layer of A^2, and provides fault-tolerant consensus with Paxos, while A^2 provides agreement with 2&3PC.

2.3 Design Rationale

In this section, we discuss why Paxos is too expensive for low-power wireless networks. We then observe that, in fact, Paxos does not require unicast communications but can be represented as a many-to-many scheme, which makes it suitable for low-power wireless networks. Finally, we explain how the broadcast-oriented wireless medium provides opportunities to develop an efficient and low-latency version of Paxos.

2.3.1 Cost of Paxos

The goal of Paxos is to provide a solution to the consensus problem that is resilient against both node failures and network faults. It does so by relying on the responses of a majority of nodes only, and by providing semantics that force all nodes to agree on the exact same decision. It is proven to be correct and has become one of the default protocols for consistent data replication within data-centers.

Paxos is an expensive protocol to run, especially in resource-constrained wireless networks. To achieve consensus, Paxos requires $2N + 2$ messages, N being the number of nodes. For example, the public and multi-hop deployment of Euratech [110] is composed of 188 nodes. Paxos would therefore need to transmit 378 messages to achieve a consensus, usually by relying on end-to-end routing.

While data-centers networks can easily sustain this cost, it becomes impractical in low-power wireless networks with strong resource constraints, as bandwidth and memory are limited, and nodes must keep their duty cycle to a minimum to save energy.

2.3.2 Paxos Beyond Unicast?

We observe that, indeed, unicasts are not needed when Paxos is executed outside of wired networks.

Paxos with multicast. A phase of Paxos can be divided into two steps: a proposer sharing its request with acceptors, and acceptors responding to the proposer. The first step can be assimilated to a broadcast, or a *dissemination*. Using unicast to disseminate a request is expensive, and several approaches use multicast communication instead (e.g., [73, 108]).

The second step can be assimilated to a *collection*, where each response might be different. Multicast cannot be used here, as Paxos requires many acceptors to communicate towards one proposer. To solve this, Ring Paxos [108] builds a logical ring composed of acceptors. An accept response traverses the ring if no higher proposal is present, and the proposal is chosen once a response traversed back to the proposer. By doing so, Ring Paxos minimizes the number of unicasts needed, but heavily relies on a stable topology.

Many-to-many. We make the observation that the majority of acceptors does not need to be statically defined in advance (e.g., by constructing a ring), but instead can be composed on the fly. By exploiting the broadcast nature of the wireless medium, acceptors (locally) broadcast their response, and *aggregate* responses heard from their neighbors. Eventually, aggregation

leads to a majority of responses combined. Additionally, we explain in §2.4.1 that a proposer is not dependent on each response individually, but rather on the application of the maximum function over those responses. We can therefore distribute this logic to all acceptors, thus completely removing the need for unicast.

The execution of Paxos therefore becomes a disseminate-and-aggregate scheme (many-to-many), which ideally maps to the concept of Chaos [42]. Fig. 2.3 shows how an execution of Wireless Paxos looks like when using concurrent transmissions and an aggregation function, in contrast to the traditional execution of Paxos (Fig. 2.1). In conclusion, it is possible to express Paxos as a many-to-many scheme, which can be efficiently executed in low-power wireless networks.

2.3.3 Basic Idea: Wireless Paxos

Based on §2.3.2, we have a mapping between Paxos and Chaos, a protocol for many-to-many communication that is highly reliable and low-latency. Therefore, it is possible to design an efficient version of Paxos for low-power wireless networks. We *co-design* Paxos with the lower layers of the network stack to provide network-wide consensus at low-latency. Specifically, we base the design of Wireless Paxos on three principles:

- **Broadcast-Oriented Communication:** We take advantage of the broadcast properties of the wireless medium to disseminate requests and collect responses efficiently from all nodes.
- **Concurrent Transmissions:** Like Chaos and A^2, we build on top of concurrent transmissions to provide low-latency and high reliability.
- **Local Computing and Aggregation:** We distribute the decision logic of the proposer to all nodes through an aggregation function to convert Paxos to a many-to-many scheme.

2.4 Designing Wireless Paxos

In this section, we dive deep into the design of Wireless Paxos. We begin by breaking down our solution. Next, we extend our concepts to its Wireless Multi-Paxos counter-part. Finally, we explain key mechanisms of our work.

2.4.1 Wireless Paxos

We present the design of Wireless Paxos, its phases and the effect of flooding on the protocol.

Failure model. Wireless Paxos uses a partially-synchronous communication-model and a non-Byzantine failure-model with crash recovery (see §2.2.2). Partial synchrony is given by the use of Synchrotron.

Flooding rounds and slots. As Wireless Paxos builds on Synchrotron, it follows a similar nomenclature, as depicted in Fig. 2.2. Wireless Paxos is executed within "flooding" rounds. A round is divided into slots. At each slot, a node either transmits, tries to receive a message, or sleeps. The node then has time for computation before the next slot starts.

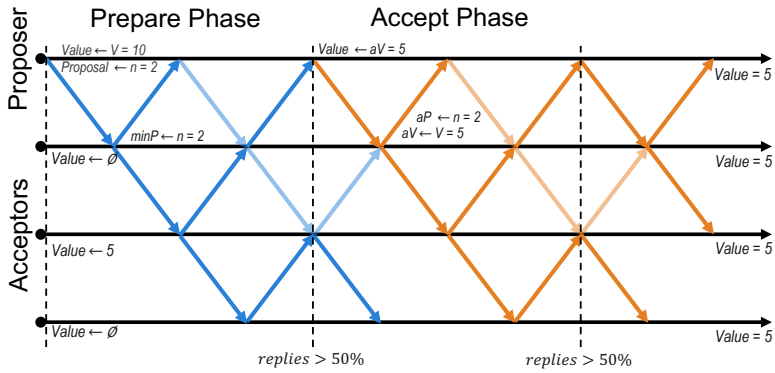


Figure 2.3. Executing Wireless Paxos: the proposer broadcasts a Prepare request. Upon reception, acceptors add their response to the message and retransmit it concurrently. Once the proposer receives a majority of participation, it switches the phase and sends an Accept request. Messages are propagated in the network and the value is adopted.

Prepare phase. In the original protocol, a proposer broadcasts its proposal number to a majority of acceptors, which reply with their accepted proposal and value if any. The proposer then selects the result with the highest proposal number and reuses the value associated.

In Wireless Paxos, the proposer starts to disseminate a prepare request with its proposal number and two additional fields for the acceptors to fill in. Upon reception, an acceptor applies the maximum function between its accepted proposal and the one received. If the local variable is higher, it replaces both the proposal and the value by the ones in memory. In addition, an acceptor always indicates its participation to the message by setting a corresponding bit in the flag field of Synchrotron.

The phase finishes as soon as the proposer receives back its proposal with at least more than half of the flags set. Applying locally the maximum function will always lead to the highest proposal number of the majority because the maximum is a commutative and idempotent function.

Accept phase. We follow a similar principle for the Accept phase. In the original protocol, the proposer broadcasts the value to agree on, as well as its proposal number. The acceptors reply with the highest proposal they prepared for. If the proposer receives a reply with any proposal higher than its own, the proposal is rejected.

In our implementation, the proposer broadcasts its proposal number, the value to agree on and an additional empty field for the responses. Upon reception, an acceptor returns the highest proposal it heard of. The proposer can therefore learn if it lost the consensus, i.e., there is a proposal higher than its own, or that the value was chosen once that a majority participated in the agreement. Again, we use the maximum function here. While it is not mandatory for the proposer to learn which proposal is the highest, this information can be used in the case of competition between multiple proposers.

Optional dissemination. Paxos guarantees that a majority of nodes, but

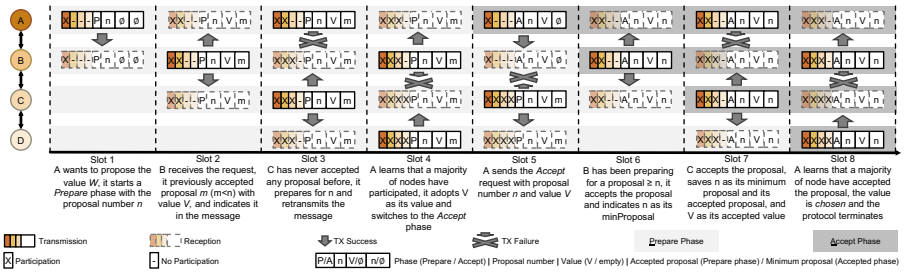


Figure 2.4. Wireless Paxos in action: all nodes act as an acceptor and node A acts additionally as a proposer. Node A wants an agreement on value W. It initiates at slot 1 a Prepare phase with proposal number n. The value and proposal fields are left empty. At slot 2, B informs the network that it has previously accepted value V with proposal number m. At slot 5, A learns that a majority of nodes have participated, it adopts value V as its own and switches to Accept phase in slot 6. At slot 7, B informs that the highest proposal it prepared for is n, and accepts the value V. At slot 8, A learns that a majority of nodes have accepted the value since no higher proposal than n was reported, the value V is therefore chosen by the network. RX failures happen due to concurrent transmissions. Transmissions continue afterwards until all nodes receive all the flags set.

not all, are aware of a value being proposed (but not chosen). It also guarantees that no other value can be shared in the network afterwards. The proposer must contact the learners to disseminate the final decision. Since our design is based on flooding, Wireless Paxos provides a built-in dissemination of the decision.

Any node receiving an Accept phase with a majority of participation and no higher proposal included knows that the value is chosen, and can *learn* this value as the final decision. In addition, the network-wide flooding of Synchrotron will force all nodes available to hear about the decision, therefore providing a network-wide dissemination of the result with high probability.

A typical execution. Fig. 2.4 represents an execution of the Paxos primitive with four nodes. All nodes act as acceptors while A is the only node with a proposer behavior. A starts the flooding round with the objective of agreeing on value K. However, node B has accepted a proposal with value V in the past. This example shows how Paxos avoids inconsistencies by adopting previously accepted values.

2.4.2 Wireless Multi-Paxos

Next, we design a primitive that provides the functionality of Multi-Paxos for agreement on a continuous stream of values and state machine replication. The optimization of Multi-Paxos over Paxos results in a lower duty-cycle, as we now only execute the Accept phase most of the time. We highlight selected mechanisms of Wireless Multi-Paxos.

Bounded memory. The original Multi-Paxos specification requires both

unbounded memory and message size. Like other implementations, we relax those requirements and limit the memory space available. A memory buffer, or log, is used to save the last values agreed upon, with a predefined *log size*. A *message length* is fixed to accommodate multiple values in one flooding round. A large log size allows nodes to recover from past failures.

Multi-Paxos allows multiple (Paxos) rounds to be executed at once, by aggregating all the requests into one message. Our design also allows to agree on multiple values at once, but requires that those rounds are consecutive in order to reduce the amount of data transmitted (see §2.2.3).

Prepare-phase specifics. Multi-Paxos requires proposers to learn the outcome of all previous rounds since the start of the system. The goal is to avoid inconsistencies and allow nodes with missing values to recover, or insert a special no-operation token to maintain state machine replication if no value was chosen. Wireless Multi-Paxos provides the same behavior. Proposers must learn all previous values in memory before agreeing on any new value.

However, a proposer will not be able to learn all previous values at once if the *message length* is smaller than the log of values. We use an *iterative learning process*, that allows the proposer to iterate back-and-forth between the prepare and accept phase until all previous values have been learned.

A new proposer starts the prepare phase to learn old values, disseminates them with the accept phase, and iterates back to the prepare phase to learn the next values, until all values have been learned.

Ordering messages. With Wireless Paxos, it is easy to compare two messages to decide which one is newer. The tuple $(proposal, phase)$ is sufficient, since a higher proposal is always newer, and the Accept phase is a newer message if both requests have the same proposal number.

However, with Wireless Multi-Paxos, the comparison of the phase does not hold anymore. We extend it with the tuple $(proposal, round, phase)$. Since the round number increases with the iteration process, messages are correctly ordered once again.

Leader lease. For the optimization to hold, Multi-Paxos requires that at most one proposer is present in the system for a prolonged period of time. The literature refers to this proposer as the *leader*. To avoid unnecessary competition between self-proclaimed leaders, we implement a *leader lease*. Each node implicitly acknowledges the proposer with the highest proposal as the leader, and promises not to compete until it believes the leader has crashed.

Once it believes the leader has crashed, a node throws a coin before proclaiming itself as the new leader.

2.4.3 System Details

In this section, we list key mechanisms at play in Wireless Paxos that do not depend on a specific phase or protocol.

Proposer and acceptor. In Wireless Paxos, all nodes in the network act as acceptors. In addition, any node can act as proposer. If this is the case, the node will first execute the acceptor logic, and then the proposer within the same slot. Due to the properties of the protocol, all nodes are also learners, at no additional computing cost.

Proposal cohabitation. We explain in §2.4.1 how to order messages.

Table 2.1. Estimating the Cost of Feedback in Euratech with 188 nodes. By repeating the flood multiple times, reliability is improved but no feedback is available (Repeated Glossy). Each node can report its status with a new flood (Glossy with Feedback) at a very high cost. Wireless Multi-Paxos provides consensus at a lower cost.

Protocol	Repeated Glossy	Glossy with Feedback	Wireless Multi-Paxos
Latency [ms]	100	3760	500

Paxos requires acceptors to discard any request older than the *minProposal* request. We supplement this requirement by transmitting the newest message every time an older request is received. We therefore reduce the risk and cost of propagating outdated data.

Phase cohabitation. Due to the flooding mechanism of Synchrotron, both phases will co-exist during a transitional period. We reduce this transition period by two means: **(a)** The proposer (and all acceptors) will always transmit the new phase if they receive an older phase transmission; and **(b)** All other nodes will reduce their transmission rate if they receive a prepare phase message with a majority of flags and resume at the normal rate once an accept phase is received.

2.4.4 Design Discussions

In this section, we explain why building consensus with Glossy is more costly than Wireless Paxos.

Cost of Feedback. Glossy offers an ultra-low latency and highly reliable ($> 99.99\%$) dissemination. It means that, under normal conditions, most of the nodes in the network will receive the value. It is however not guaranteed, since Glossy does not use explicit feedback. For example, an initiator will never detect a network segmentation, or high message losses, or node failures. An initiator only knows if at least one node received the flood (due to the semantics of Glossy), and nodes implicitly detect missed floods, but cannot recover the value. Repeating multiple times the flood increases further reliability, but still doesn't provide additional guarantees. To provide explicit feedback, each node would be required to start its own flood to report its status (e.g., see [29]).

Small example. For example, in the Euratech testbed, with $N = 188$ nodes, a flood takes 20 ms. Table 2.1 gives a back-of-the-envelope calculation of the cost of feedback with Glossy. Repeating 5 times a flood takes 100 ms. Receiving feedback from all nodes requires 3760 ms. In contrast, we show in §2.5.3 that 193 ms are needed with Wireless Multi-Paxos to get a majority of replies, and 500 ms for all nodes to hear about all other nodes.

2.4.5 On the Correctness of Wireless Paxos

In this section, we give a short and informal walk-through of why Wireless Paxos does not break the correctness of Paxos (see [52] for the original proofs).

Table 2.2. Statistics and parameters of testbeds used in the evaluation. They represent a very dense and a low-density deployment, respectively. Both are collocated with Wi-Fi and Bluetooth deployments.

Testbed	*Size* [#]	*Coord.* ID	*Dens.* [#]	*Diam.* [hops]	*Chann.* [#]	*TX Pw.* [dBm]
Euratech	188	3	106	2	16	0
Flocklab	27	3	7	4	2	0

Table 2.3. Slot length of each protocol. Due to their complexity, WPaxos and WMulti-Paxos require more computation time.

Protocol	Glossy, 2&3PC	WPaxos	WMulti-Paxos
Slot Length	4 ms	5 ms	6 ms

The main difference between Paxos and its wireless counterpart lies in the *maximum* function being distributed from the proposer to all nodes. While Paxos requires a majority of replies, Wireless Paxos requires an aggregate from these replies.

As both the *maximum* and the *(set) union* functions — used to compute the number of replies — are commutative and idempotent, the local execution by all nodes of both functions is equivalent to the execution of the functions by the proposer alone. As such, Wireless Paxos maintains the safety properties of Paxos.

2.5 Evaluation

Paxos is a notoriously complex protocol to implement, and even tougher to evaluate [102,103]. Wireless Paxos is no different from that perspective. Proving that an implementation of more than a thousand lines is correct is often an extremely delicate and cumbersome task. We instead decide to use an extensive testing approach (like [103]), during which we evaluate our system on physical deployments of different topology and density.

We begin by discussing our setup. We then follow a bottom-up approach, and present how a node's state evolves during a round. Then, we compare both the cost and fault resilience of Wireless Paxos with related network-wide primitives.

2.5.1 Evaluation Setup

In this section, we lay out how we implement Wireless Paxos and what scenarios are executed for the evaluation. We then list the different metrics and testbeds used.

Implementation. We implement Wireless Paxos in C, on top of Synchrotron [75], for the Contiki OS. We target wireless sensor nodes equipped with a low-power radio such as TelosB and WSN430 platforms which feature a 16bit MSP430 CPU @ 4 MHz, 10 kB of RAM, 48 kB of firmware storage and a CC2420 [111] radio compatible with 802.15.4.

Testbeds. We use two publicly available testbeds for our evaluation: Flocklab [112] and the Euratech deployment of FIT-IoT Lab [110]. Flocklab is composed of 27 nodes while Euratech contained up to 214 nodes, with around 188 active during our evaluation. Table 2.2 summarizes properties of both deployments. Due to the closure of the Euratech testbed, some parts of the evaluation are carried in Flocklab only.

Scenarios. We evaluate the following applications:

- The Wireless Paxos primitive (*WPaxos*): one proposer starts a consensus on a 1-byte data item, leading to a total payload of 31 bytes in Euratech and 10 bytes in Flocklab. Both phases are executed at every flooding round. We refer the reader to [42] for the effect of varying payload size over the system;
- The Wireless Multi-Paxos primitive (*WMulti-Paxos*): one proposer starts a series of consensus on 1-byte data items. Each flooding round corresponds to one value. Acceptors keep a log of the last four values they agreed on. The Prepare phase is executed during the first flooding round and is skipped afterward;
- WPaxos with multiple proposers: a pre-defined number of proposers are competing at each flooding round on different data items;
- Consistency: At each slot, a node has a pre-defined probability to enter in failure mode. A failed node stops to use its radio. At the end of a flooding round, we compare each node's state to detect any inconsistency in the consensus.

Metrics. We focus on the following indicators:

- Progress and State: the progress of a node represents the number of participants that node heard of. Its state represents the phase of the protocol (prepare, accept);
- Latency is refined into two notions: the *Paxos latency*, representing the time when a value is chosen, and the *full completion latency*, representing the time when the lower layer Synchrotron converged (see §2.4.1 and §2.5.3);
- Radio-on time: the total time the radio is active during a flooding round. It is used as a proxy for the energy consumed during a round;
- Consistency: from a network-wide perspective, it represents if nodes agree on the same final value.

Slot Length. Different protocols have different complexity, and require a different amount of execution time. Table 2.3 contains the different slot length used during the evaluation. Due to their complexity, both WPaxos and WMulti-Paxos require additional computation time compared to the literature.

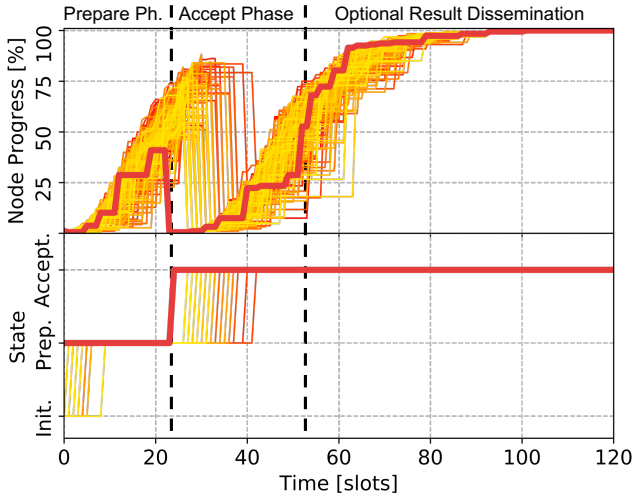


Figure 2.5. A snapshot of a typical WPaxos round: the upper figure represents the progress of each node (proposer is the thick red line) while the lower represents their state. It takes 21 slots for the proposer to receive a majority of replies and switch to the Accept phase. It takes an additional 35 slots for the Accept phase to complete with a majority of flags. Results converged after 115 slots and the flood finishes.

Note that the hardware used features a 4 MHz CPU. The computation time would be reduced on newer hardware with a faster CPU.

2.5.2 Dissecting Wireless Paxos

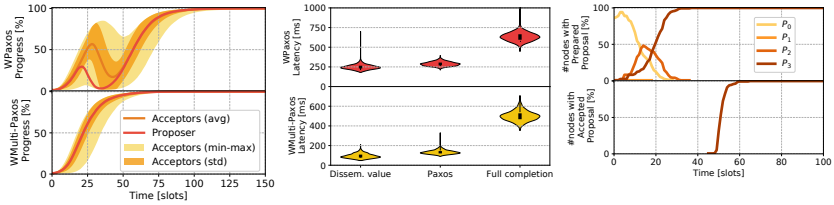
We evaluate how a consensus is handled with Wireless Paxos. We first analyze a representative instance of the protocol through the different node states. Then, we compare the execution of WPaxos and WMulti-Paxos.

Basic Round. Fig. 2.5 depicts a representative execution of Wireless Paxos in Euratech with 188 nodes. All nodes start in the initial, empty state.

At slot 0, the proposer (represented by a red thick line) starts the round with a Prepare request. Since the proposer also acts as an acceptor, the node transitions into the Prepared state. Due to the high density of Euratech, it takes around 10 slots (50 ms), for all nodes to hear the request and enter the prepared state.

After 21 slots (105 ms), the proposer detects that a majority of nodes participated and replied with a promise. It starts the accept phase, resets back the progress to 0 and switches into the Accepted state. Due to the prepare phase still spreading, a transition period of roughly 20 slots (100 ms) is necessary for the acceptors to learn about the new phase. At slot 56 (280 ms), the proposer learns that a majority of the nodes accepted the value (referred as *Paxos latency*). The value is therefore *chosen* and the consensus succeeded.

The results naturally converge and at slot 115 (575 ms), the 188 nodes learned that the entire network agreed on the value (*full completion latency*).



(a) Comparing WPaxos and WMulti-Paxos. **(b) Different Definitions of Consensus.** **(c) Multiple Proposers Competing (WPaxos).**

Figure 2.6. Executing Wireless Paxos (WPaxos) and Wireless Multi-Paxos (WMulti-Paxos) in Euratech with 188 nodes. ***(a)*** **WMulti-Paxos requires only one phase and finishes in roughly 83 slots, while WPaxos requires around 127 slots for its two phases.** ***(b)*** **In Euratech, 248 ms and 94 ms are necessary for a dissemination to all 188 nodes for WPaxos and WMulti-Paxos respectively. 289 ms and 133 ms are needed for the proposer to hear a majority, and it takes 633 ms and 500 ms for a network-wide knowledge of the result.** ***(c)*** **A line represents the number of nodes that locally prepared or accepted a proposal, but not the progress seen by a proposer. Proposals are initially competing but quickly ruled out by the highest proposal, and only P_3 sees a majority of replies.**

Note. It takes exactly the same time if another value is present in the system. The proposer would simply replace its own value by the one reported at the end of the first phase, and disseminate it during the accept phase.

WPaxos and WMulti-Paxos. Paxos is not an efficient protocol. The prepare phase is necessary only if multiple proposers are present or the node just started as proposer. The first phase is therefore superfluous the rest of the time. Multi-Paxos builds on that knowledge and executes the prepare phase only once (cf. §2.2.3 and §2.4.2).

Fig. 2.6a compares the average *full completion latency*, i.e., time until Synchrotron completion, for WPaxos and WMulti-Paxos with 188 nodes. The red line represents the proposer's progress, while the orange line represents the average progress of the acceptors. The dark-orange area represents the standard deviation of the acceptors' progress, and the light-orange area the minimum and maximum progress, i.e., the slowest and fastest acceptor respectively.

It takes by average 127 slots (633 ms), for WPaxos to complete (from a Synchrotron perspective), while it takes only 83 slots (500 ms) for WMulti-Paxos. Removing the first phase improves the latency although WMulti-Paxos requires more computation time (cf. Table 2.3).

2.5.3 Paxos and Primitive Latencies

Paxos defines a consensus complete once a majority of accept responses have been received by the proposer. However, the lower layer Synchrotron does not stop communication at that point, but continues until all nodes have converged (i.e., until all nodes see all flags set, see §2.4.1). We measure the latency from

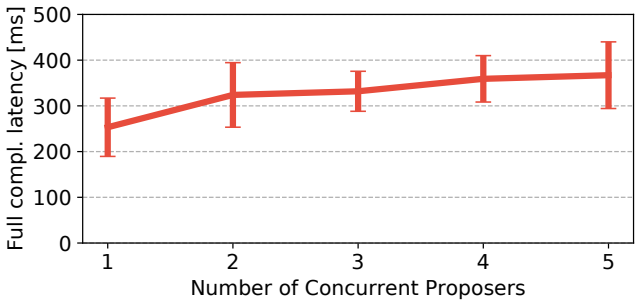


Figure 2.7. Cost of Multiple Proposers in Flocklab: the competition between proposers causes a latency overhead. After 3 concurrent proposers, the overhead stops growing.

Paxos and Wireless Paxos perspectives.

Metrics. We select three definitions of latency: **(a)** *Disseminated value*: similar to Glossy, corresponds to the time required for all nodes to hear the value the first time; **(b)** *Paxos*: following Paxos definition, a correct consensus requires that the proposer receives a majority of replies during the accept phase; **(c)** *Full completion*: similar to the *Max* primitive of Chaos [42], the latency is defined as the time required for all nodes to hear from all other nodes.

Results. Fig. 2.6b shows the reported latencies. Simply disseminating the value is fast, it takes WPaxos only 248 ms to share the result to all nodes. Collecting feedback is more expensive, and it takes 289 ms for the proposer to learn that at least a majority replied to its Accept request (Paxos definition of consensus). Collecting the flags from all the nodes induces a large overhead, with a mean latency of 633 ms. On the other hand, WPaxos thus ensures that all nodes received the decision, beyond the majority ensured by Paxos.

WMulti-Paxos is faster. It takes 94 ms for the dissemination, 133 ms to hear from a majority and 500 ms to receive all flags.

2.5.4 Influence of Multiple Proposers

In this section, we study the effect of having more than one proposer in the system.

Scenario. Traditional agreement protocols require at most one node to act as a leader, and fail if multiple are present. Paxos, on the other hand, can deal with the presence of more than one proposer in the system. We run several hundred WPaxos rounds while increasing the number of proposers in the system. Proposers are chosen following a uniform distribution, and send a prepare request as soon as a lower proposal is received. A proposer stops competing once a packet with a higher proposal number is received.

In a round. Fig. 2.6c depicts the competition between four proposers in the highly dense deployment of Euratech. P_0 represents the proposal of the flood initiator, while P_1 to P_3 are from randomly chosen proposers. As the request floods the network, many acceptors prepare for P_0. The other proposals start competing very early but their dispersion is slower due to the collisions during transmission. Intermediate proposals P_1 and P_2 cannot gain enough

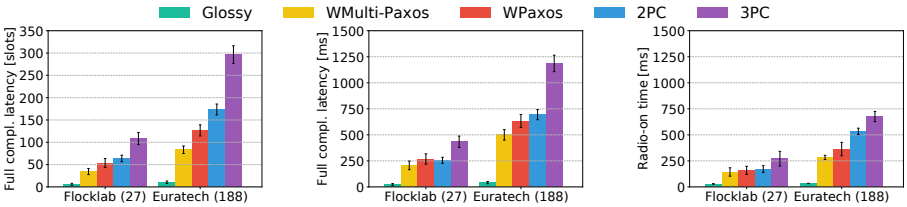


Figure 2.8. Comparing the cost of different primitives: Glossy solves dissemination, WPaxos and WMulti-Paxos solve consensus while 2PC and 3PC (here from A² [75]) solve commit (agreement). The cost increases with the complexity (i.e., the number of phases). The latency is measured for the network-wide dissemination of the final decision.

momentum to collect a majority of replies.

While both P_0 and P_3 are heard by a majority of acceptors, only P_3 manages to collect enough responses and "wins" the competition. After roughly 50 slots (250 ms), the proposer of P_3 starts disseminating the accept phase. The proposal is accepted as it propagates through the network.

Latency. Fig. 2.7 characterizes the effect of competition in terms of full completion latency in Flocklab, although the Paxos latency follows the same behavior. The presence of multiple proposers induces an overhead of up to 100 ms in Flocklab.

As the number of concurrent proposers grows, the overhead starts to plateau. Acceptors quickly discard lower proposals to force the spread of the highest proposal, and competitors are quickly ruled out of the system. In addition, a proposer will not compete if its own proposal is lower than the proposal received.

With WMulti-Paxos, the overhead is only present when the first phase is executed and with similar results. Any consecutive round is executed as usual as long as proposers do not compete again.

2.5.5 Comparing the Cost of Primitives

We evaluate the cost of running the WPaxos primitive, and compare it with dissemination (Glossy [23]) and agreement primitives (2&3PC from A² [75]), in terms of latency and radio-on time. Since the lower layer Synchrotron continues to communicate after a value is chosen, we use the full completion latency (see §2.5.3) as metric.

Scenario. We compare five applications: **(a)** Glossy Mode (Glossy), a one-to-all dissemination mode without flags; **(b)** Two-Phase Commit (2PC), an all-to-all agreement protocol with flags and votes introduced to A²; **(c)** Three-Phase Commit (3PC), similar to 2PC but with an additional phase, also introduced by A²; **(d)** the Wireless Paxos primitive; and **(e)** the Wireless Multi-Paxos primitive.

Experiments are run both for Flocklab with 27 nodes and Euratech with 188 nodes. All applications are executed for 2500 rounds in Flocklab, and for a thousand rounds in Euratech. Glossy, 2PC and 3PC are executed only for several hundred rounds in Euratech due to the final closure of the testbed. The

results reported here are nonetheless consistent with the results reported by A^2 [75]. The slot length used are summarized in Table 2.3.

Results. Fig. 2.8 summarizes the comparison in terms of full completion latency, both in slots and milliseconds, and in terms of radio-on time. First, we observe a cost increase with the number of nodes. The increase is however not proportional, as Euratech is seven times larger than Flocklab but induces an increase of roughly 2.5× only. This is mainly due to the difference in density and topology of the deployments.

We now take a look at the performance of each application. Glossy is the fastest since it does not require feedback, but does not provide the same guarantees as consensus (see §2.2.1). WMulti-Paxos has the second lowest cost, since only one phase is executed. WPaxos shows a cost of roughly 1.5× the cost of WMulti-Paxos in terms of slots. This is due to the fact that the primitive has two phases, but the first one requires half the nodes only. 2PC is roughly 2× more expensive than WMulti-Paxos in terms of slots, while 3PC is roughly 3× more costly. Again, it translates to the two and three phases of 2&3PC, respectively.

Due to the different slot length (cf. Table 2.3), the improvements of WMulti-Paxos are less visible when considering the latency in milliseconds. WPaxos takes 1.3× the time of WMulti-Paxos, while 2PC takes 1.4× the time in Euratech and 3PC takes 2.4× the time of WMulti-Paxos. Again, we point out that these results are due to the hardware used (4 MHz CPU). Modern hardware would provide improved results, closer to the slot latency.

Note that in Flocklab, WPaxos and 2PC presents a similar latency, even if WPaxos requires fewer slots. For small deployments, the length of the slot has a bigger effect than for dense deployments.

Finally, the radio-on time represents how long the radio was active (either transmitting or receiving), and is used as a proxy for energy consumption. Once again, WMulti-Paxos and WPaxos require fewer transmissions than their counterparts 2PC and 3PC, since they rely on majorities.

2.5.6 Primitives Consistency

In this section, we evaluate the consistency of the different primitives under injected failures.

Scenario. Paxos, by design, is tolerant against failures, and solve them by using majorities, while 2PC delays the decision (consistency over availability). We compare how the different consensus primitives are affected by node failures.

We run Wireless Paxos on Flocklab for 900 rounds with different failure rates. At each slot, each node can fail following a given probability (from 0 to 4×10^{-5}), i.e., it stops communicating for the duration of the round. This failure model is similar to a network segmentation or a crash-recovery where the node saves the result of the consensus in a stable storage.

A round is considered consistent if all nodes have the same value in the end. In WPaxos, a consensus is also consistent if at least a majority share the same decision, even if some node missed the value. We refer to those cases as *MAJ-consistent*. A system can also abort a decision in 2PC and 3PC. The result is consistent if all nodes have aborted. A blocked round means no decision has been chosen yet. Other cases are inconsistent.

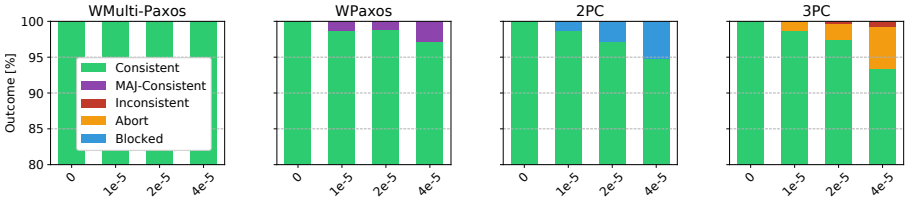


Figure 2.9. Consensus consistency under injected failure: WPaxos handles failures with majorities and semantics, while 2PC is blocking (consistency over availability), and 3PC is non-blocking but sometimes inconsistent.

Results. Fig. 2.9 shows the result of consensus under failure. Because WMulti-Paxos executes the accept phase only, the consensus is faster and less prone to failure. WPaxos, with its two phases, is slightly less resilient. As the failure rate increases, some nodes are missing the final value. The consensus remains nonetheless correct and nodes can eventually learn the decision, keeping the system consistent.

2PC blocks if at least one node is missing the decision. Strong consistency is maintained at the cost of availability. 3PC requires more communication, and is thus more prone to faults. Some rounds were inconsistent (some nodes aborted while others committed). However, 3PC is non-blocking by design.

2.6 Related Work

Concurrent Transmissions. Glossy [23] is one of the pioneer works in the field of concurrent transmissions. In Glossy, nodes synchronously transmit the same packet, allowing constructively interfering signals to be received. Glossy thus provides highly reliable and low latency dissemination. LWB [27] and Crystal [28] use Glossy to provide data dissemination and collection from all nodes by scheduling and executing floods sequentially.

By relying on the capture effect instead of constructive interference, Chaos [42] and Mixer [43] relax the tight synchronization condition and can transmit *different* data concurrently. Chaos uses in-network processing to provide data collection and aggregation by applying an aggregation function locally over the received data, while Mixer uses network coding to provide many-to-all communication.

Consensus. In distributed systems, agreement and consensus have been extensively studied for several decades already. Well-known solutions include 2PC [49] and 3PC [50]. Other solutions, like PBFT [113], provide a solution in the presence of Byzantine faults. Paxos is a general (non-Byzantine) fault-tolerant solution to the consensus problem [51, 52]. Paxos has been extended and further optimized, for example with Fast Paxos [114], Cheap Paxos [115] or Ring Paxos [108]. Raft [102] is an alternative to Paxos, designed to be more comprehensive.

Consensus in WSNs. Consensus has been studied in opportunistic and ad-hoc networks [70, 71]. However, most approaches focus on single-hop networks [74, 116]. Consensus with Byzantine failures in single-hop networks

has also been studied [72].

Most multi-hop consensus solutions rely on routing. Köpke proposes an adapted 2PC for WSNs [26], while Borran et al. extends Paxos with a new communication layer for opportunistic networks [73]. Borran's solution relies on the MAC layer of 802.11 and builds a tree to collect and route responses. Furthermore, unicast and acknowledgments are used for collecting responses. In contrast, this work co-designs Paxos with the lower layers of the network stack to provide an efficient and low-latency consensus primitive for low-power wireless networks. We do not rely on any routing, but utilize concurrent transmissions to communicate in multi-hop networks.

A^2 provides an implementation of 2&3PC using concurrent transmissions [75]. Wireless Paxos reuses the transmission kernel introduced by A^2, but the consensus primitives differ. A^2 handles failures by delaying the decision (consistency over availability), while Paxos handles failures through majorities.

Finally, VIRTUS [29] brings virtual synchrony to low-power wireless networks. Virtual synchrony provides atomic multicast, i.e., it allows to deliver messages in order to all members of a group, or none. Virtual synchrony and Multi-Paxos are two ways to create state machine replication in distributed systems. Both solutions provide guarantees on the consistency of delivered messages. We argue in §2.4.4 why Glossy-based schemes (like VIRTUS) induce higher costs than Wireless Paxos.

2.7 Conclusion

This paper presents Wireless Paxos, a fault-tolerant, network-wide consensus primitive that builds on top of concurrent transmissions to offer low-latency and reliable consensus in low-power wireless networks. We argue that although established consensus protocols like Paxos offer many benefits like fault-tolerance, correctness, and consistency guarantees, their designs, based on unicast communications, make them unfit for low-power wireless deployments. Wireless Paxos fills this gap by showing that Paxos can be expressed as a many-to-many communication scheme, and by co-designing the consensus primitive along with the lower layers of the network stack and concurrent transmissions to offer highly reliable and low-latency consensus. We experimentally demonstrate that Wireless Paxos **(a)** guarantees that at most one value can be agreed upon, **(b)** provides consensus between 188 nodes in a testbed in 289 ms, and **(c)** stays consistent under injected failures.

2.8 Acknowledgments

We would like to thank the anonymous reviewers, as well as Simon Duquennoy, Marco Zimmerling, and Carlo Alberto Boano, for their valuable comments and suggestions to improve the paper. This work was supported by the Swedish Foundation for Strategic Research (SSF) through the project LoWi, reference FFL15-0062.

3

STARC: Low-power Decentralized Coordination Primitive for Vehicular Ad-hoc Networks

---

P. Rathje, **V. Poirot**, O. Landsiedel

Third International Workshop on Intelligent Transportation and Connected Vehicles Technologies (ITCVT), IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–6.

PAPER B

Abstract

Intersections are the bottlenecks of road networks. Coordination mechanisms for intersection crossing greatly affect the efficiency of road utilization. Typically, coordination is done by implanting local infrastructure, whether signs, traffic lights, or through common, well known-rules shared by all users. In this paper, we introduce STARC, a decentralized intersection management protocol for future connected vehicles and other traffic participants. With STARC, all participants coordinate their movement using reservations to guarantee safe crossings. To enable cost-efficient deployment, STARC does neither rely on any centralized infrastructure, such as traffic lights, nor centralized wireless intersection coordinators, like virtual traffic lights. STARC targets small, cheap, and energy-efficient platforms and the open low-power wireless standard 802.15.4 so that all participants in road traffic could take advantage of it, including vehicles, bikes, electric scooters, and even pedestrians. STARC builds on low-power wireless communication with A^2-Synchrotron and multi-hop routing as a communication substrate and provides distributed transaction, election, and handover mechanisms to manage the intersection cooperatively. We show that STARC reduces average waiting times by up to 50% compared to a fixed traffic light schedule in traffic volumes with less than 1000 vehicles per hour. Moreover, we illustrate a platoon extension that allows STARC to outperform traffic lights even at traffic loads beyond 1000 vehicles per hour.

3.1 Introduction

Intelligent Transportation Systems (ITS) lay the foundations for new, efficient ways of mobility. By adopting ITS, traffic delay, fuel consumption, and greenhouse gas emissions could be reduced [24, 79]. At the same time, vehicle-to-everything (V2X) communication could replace infrastructures like traffic signs and lights and minimize infrastructure costs [78, 79]. Protocols in this domain often rely on the presence of cellular network coverage or central servers for coordination [24, 79]. In contrast, decentralized approaches do not require additional infrastructure as they solely rely on communication between the vehicles [15, 117]. However, most of these focus on bandwidth intense technologies such as 802.11p or 5G device-to-device (D2D) and are inherently limited to platforms with sufficient compute and energy resources. Moreover, many proposals neglect a detailed performance and reliability evaluation in realistic communication environments [118].

In contrast, we argue that there is a need for distributed, safe, and efficient protocols for traffic coordination able to run on resource-constrained platforms.

Challenges. Algorithms for robust and fault-tolerant coordination have an inherent complexity that defines a stark contrast to the resource-constrained low-power wireless platforms we target. Thus, we need to devise algorithms and a system design that can deal with unreliable wireless communication, complex routing, and group-membership in a delay-sensitive environment — without sacrificing efficiency and safety.

Approach. We introduce STARC, a decentralized reservation-based protocol that uses cheap, low-power wireless radios to enable energy-efficient vehicle-to-vehicle communication. We build our coordination protocol on top of A^2-Synchrotron (Synchrotron), a low-latency and energy-efficient communication primitive for all-to-all communication [75]. With STARC, traffic participants reserve lanes to cross the intersection. We provide transaction semantics, and all participants coordinate to commit on a shared access pattern jointly. As a result, vehicles have unique access to different parts, i.e., lanes, of the intersection, ensuring that at most, one car can use a given lane. Once a car has crossed the intersection, it releases its reservation and leaves the communication network, allowing the next car to continue its journey.

Our design and implementation for IEEE 802.15.4 radios allow STARC to operate on energy-restricted devices to support all road users, including cyclists and pedestrians.

Contributions. This paper contributes the following:

- We present STARC, a decentralized protocol for autonomous intersection management.
- STARC provides fault-tolerant, distributed coordination, and transactions in dynamic networks.
- Targeting all traffic participants, we design and implement STARC for resource-constrained IoT platforms.
- We evaluate STARC on a simulated intersection showing its efficiency and fault-tolerance under injected interference and radio failures.

Outline. After we cover background, related work and an introduction to Synchrotron in Section 3.2, we present the design in Section 3.3 and its evaluation in Section 3.4. Finally, we conclude our work in Section 3.5.

3.2 Background

Intelligent Transportation Systems build on computer systems to aid traffic optimization. Brake assistants, crash, and congestion warnings are example applications. Vehicles may also form platoons to save fuel and increase road utilization [119]. Such advanced intents require communication of the vehicles with each other (V2V) or with the infrastructure (V2I) [78]. Available protocols for V2V and V2I communication are based on IEEE 802.11p (WAVE) or cellular protocols such as LTE or 5G [120, 121]. In Section 3.2.1, we review current solutions for intersection coordination. Then, in Section 3.2.2, we introduce our communication substrate, Synchrotron.

3.2.1 Related Work

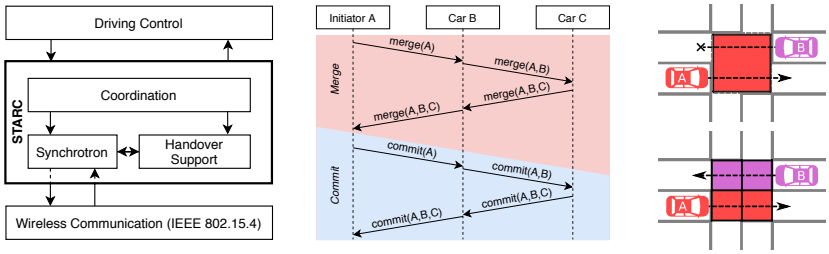
Adams and Rutherford show the potential of decentralized intersection management [122]. They compare two peer-to-peer algorithms with the centralized approach by Dresner and Stone (AIM) [79].

Vanmiddlesworth et al. present a reservation coordination-protocol based on claims [117]. In their approach, cars broadcast claims for the intersection repeatedly, and the one with the most dominant claim may proceed. While they show that their approach doubles the throughput compared to ordinary stop signs in low volume scenarios, the protocol itself can not handle communication failures safely [123].

Virtual Traffic Lights (VTL) replace physical, centralized traffic lights with a virtual coordination infrastructure by sending messages between the participants [15]. In case of potential conflicting directions, vehicles elect a single vehicle as the coordinator, which generates and broadcasts a schedule. As soon as the coordinator is allowed to drive, it will handover coordination to another vehicle. Ferreira et al. show that Virtual Traffic Lights can reduce CO_2 emissions by up to 18% [124]. VTL are shown to improve the driving experience as they reduce mean travel and waiting time [125]. Further optimizations are made by Sommer et al. by dynamically changing the phase length based on the number of waiting cars [123].

The protocol of Hassan and Rakha reduces the number of messages sent by letting only the leading vehicles of each lane create the schedule [126]. Each leading vehicle then propagates the schedule back through the lanes. Naumann et al. propose a semaphore-based algorithm which only allows one vehicle to stay in each critical region of an intersection [127].

In contrast, STARC builds its crossing schedule via a 2-Phase Commit-like network-wide agreement. Since all participants must participate and agree for a schedule to be valid, STARC achieves fairness and safety: All participants eventually cross the intersection, and at most, one schedule can be selected at any instant. Moreover, we implement STARC for low-power wireless radios. Thus, any road user can take advantage of the protocol.



(a) Architecture Overview **(b) Coordination Round** **(c) Granularity**

Figure 3.1. ***(a)*** **The STARC protocol builds coordination on top of Synchrotron and adds support for join, and leave as well as the election and hand-over of a leader.** ***(b)*** **The leader starts a coordination round in the merge phase. As soon as all vehicles marked their participation, the commit phase is started and the merged schedule established.** ***(c)*** **A granularity of 1 (top) supports only a single vehicle while a granularity of 2 (bottom) enables parallel crossings.**

3.2.2 Synchronous Communication with Synchrotron

A^2-Synchrotron (Synchrotron) [75] is a wireless communication primitive combining all-to-all data sharing and in-network processing to enable interaction between all nodes in low-power wireless networks. Synchrotron provides multi-hop communication and inherently supports mobile environments. Moreover, it readily integrates channel-hopping and cryptographic primitives to ensure a robust protocol. At its heart, it makes use of two fundamental techniques: Synchronized transmissions and data aggregation.

Synchronized Transmissions. In Synchrotron, the nodes communicate in rounds. Each round is divided into multiple slots in which nodes try to receive a packet or transmit. Although multiple nodes might transmit concurrently during the same slot, some nodes can correctly receive and decode packets due to the capture effect [42]. Between two rounds, the radio is turned off to save energy.

Data Aggregation. Upon receiving a packet, a node processes the information using an application-specific merge-operator, marks its participation, and disseminates the result. This approach combines data collection, processing, and dissemination at once.

3.3 Design

Overview. The STARC protocol guarantees safe intersection crossing without the need for central infrastructure. We rely only on local, low-power wireless communication. We divide the protocol into two parts (1) Reservation-based movement coordination and (2) handover support; see Figure 3.1a.

System Assumptions. We assume that all road users are equipped with IEEE 802.15.4 radios. Moreover, we assume the presence of position measurement devices, e.g., some sensors and a unique vehicle identifier (ID).

Furthermore, we assume access to information about the intersections, i.e., lane-layout, size, and possible directions. All vehicles have (limited) computing capabilities and can plan a trajectory they want to perform. Another assumption is non-byzantine behavior: All participants are honest and follow the protocol — no vehicle purposely tries to disrupt it.

3.3.1 Distributed Reservation Coordination

Intersection setup. We divide the intersection into a $n \times n, n \in \mathbb{N}_+$ grid of even tiles [79, 128]. Each tile covers a specific area and possesses a unique address. The tile-layout of the intersection is known to every vehicle.

Tile reservation. A vehicle requests a reservation for tiles based on their addresses. Besides the tiles, a request contains the unique identifier of the vehicle and its priority. We base the priority on the arrival time and enable fairness by priority [127]: A later arrival time results in a lower priority.

Once a vehicle desires to pass the intersection, it plans its path over the intersection and tries to reserve all the tiles on its way, see Figure 3.1c. No notion of crossing time is used as we have to ensure safety also when vehicles take longer to pass the intersection as initially planned. The vehicle may only start when all needed tiles are reserved [127]. When the vehicle is leaving the intersection, it frees its reserved tiles, so that other vehicles may reserve them.

Our protocol builds on top of Synchrotron since it is designed explicitly for simultaneous data sharing and processing. We use this property to create a Synchrotron round for parallelly collecting and merging the reservations. This coordination round is split into two phases, as depicted in Figure 3.1b:

1. *Merge phase*: Individual reservation requests are collected and merged into a feasible schedule.

2. *Commit phase*: If all members participated, this phase disseminates the resulting intersection-wide schedule to ensure consensus.

Merge phase. The merge phase uses a custom merge-operator that simultaneously merges and solves occurring conflicts according to individual priorities. Additionally, the operator is order-invariant and creates partial schedules of the participated vehicles. A leader node starts the merge phase (and thus the coordination round) with the initial transmission. The progress is tracked using participation flags in the packet. With the participation of all members in the network, the schedule and thus the merge phase is complete. The reservation grid assigns the vehicle ID with the highest priority (if any) to each tile. Conflicts are resolved for each tile individually according to the assigned priorities.

After collecting and merging the individual reservation requests, the vehicles have to commit to the very same reservation-grid: The final result has to be consistent. Though the participation flags indicate that every node participated in the merging phase at some earlier point, it is not guaranteed that they are all aware of the final schedule; an unmerged request could override some parts. For this reason, we incorporate a second, commit phase. Overall, these transaction semantics are the foundation for safety in STARC: a vehicle can only cross if all others have agreed to the path.

Commit phase. The leader starts the commit phase as soon as it receives the complete schedule. It marks the packet as a commit phase transmission and clears all participation flags before transmitting it: The leader commits on the intersection-wide schedule. Further changes are not allowed in the commit phase.

When a node currently in the merge phase receives a commit packet, it switches to the new commit phase and adopts the received reservation-grid, ignoring any local, incomplete state. Its participation flag is set once more, this time as a simple acknowledgment of having received the commit. Nodes retransmit the packet according to the underlying Synchrotron mechanism using the flags as a progress indicator. The commit phase ensures that every node has the chance to participate in the chosen, unique schedule. Due to the retransmissions, almost all are aware of the schedule at the end of the round.

Crossing schedule. Vehicles can check the latest state they received at the end of the round. If it contains a commit, the vehicle verifies the state of its reservation request. The request is only accepted if all tiles along the path have been granted to that vehicle. It follows that every other request either tried to reserve other tiles or had a lower priority. Because the second phase is only started, if all participated, the commit ensures that all requests are merged. They agree on a common intersection-wide schedule.

If a vehicle got accepted, it might mark its reservation as passing for all following rounds. Such a reservation has the highest priority and wins in all conflicts in subsequent rounds. This way, the reservation grid contains both the reservations of the passing and waiting vehicles. Conflicts between passing reservations are impossible as long as the vehicles do not add new tiles to accepted reservations, which is prohibited.

When the vehicle has passed the intersection, it just empties its following requests, and others can reserve those tiles again (the protocol allows freeing unneeded tiles). While issuing more requests seems a little excessive, this mechanism does not require the vehicles to store any states of other vehicles' requests. They could even reconstruct their own state from the position in the intersection. Since there is no need to save the states of the reservations, failures can be handled well, making this part of the protocol fail-safe.

3.3.2 Handover Support and Leader Election

Before a car may participate in a coordination round, it needs to join the network. Its participation is expected as long as the car has joined the network. The car thus needs to leave explicitly. If the car drives off without leaving the network, the commit phase does not start, and the protocol would halt so that progress is impossible.

Leader handover. The leader manages the mapping of Synchrotron indices to vehicle IDs and thus determines which nodes participate with which Synchrotron index. The leader handles joins and leaves and hands-over to another vehicle as soon as it wants to leave the network. As the mapping of the joined nodes is part of that handover, and the payload of a Synchrotron packet is restricted, we limit the network size. As a result, we only allow the first cars of each lane to join the network and assume cars use their sensors to detect cars in front of them.

Join and leave. Nodes may join and leave the STARC network dynamically. A new vehicle starts listening for packets, and with the beginning of the next round, it acts as a forwarder: The node transmits according to the STARC policy and merges partial schedules but is not allowed to issue its own reservation requests until successfully joined. As soon as the vehicle passes the intersection, it leaves the network.

We incorporate the join and leave mechanisms into the coordination round and use monotonically increasing configuration numbers [75]. The nodes compare their locally saved numbers to detect inconsistencies, e.g., missed commits. Upon missed commits, a node rejoins the network to recover.

3.3.3 Platoon Extension

Vehicles that share a common direction may group up and form platoons to utilize the road better and, in our case, the intersection. We can easily extend the STARC primitive to support platoons: While waiting, the vehicles in each lane form a platoon and their platoon head coordinates and reserves the path for the full platoon as if they were a single, long vehicle. The reserved tiles are freed after the last platoon member passes it.

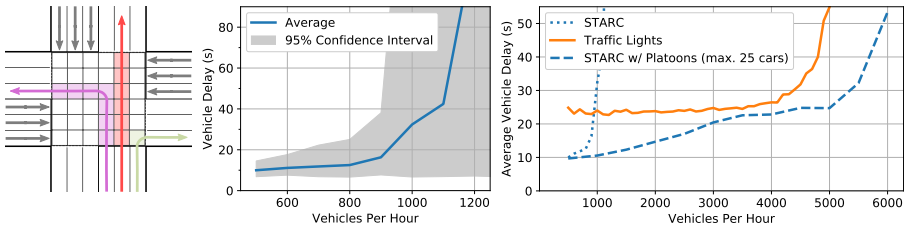
3.4 Evaluation

This section analyzes STARC's behavior in demanding wireless environments as well as its efficiency in potential traffic scenarios.

Methodology. The measurements are based on data gained through three 30 minutes simulation runs. Each vehicle performs the following steps: queueing, waiting for its reservation, moving over the intersection, and leaving the network. The times for queueing, waiting, and leaving determine the delay of the vehicle. We quantify efficiency by the average additional delay and safety by the number of collisions [79]. The fundamental parameter is the number of vehicles per hour specifying the level of traffic.

Setup. With 16 supported nodes, the network has enough slots to cover all 12 lanes (three lanes per direction) plus four additional slots for crossing or leaving vehicles. Figure 3.2a displays the simulated intersection with the corresponding tile-grid. We set the granularity to 6, the resulting tile borders thus match the borders of the lanes. The implementation features four join slots and a single rejoin slot. We set the length of a Synchrotron slot to 6 milliseconds and the number of slots to 200. The interval between two Synchrotron rounds is 2 seconds.

The cars themselves are homogenous and share the same physic specifications, as displayed in Table 3.1. The turning rate is limited to 90 degrees per second. Sharp curves thus require deceleration. As assumed in the design chapter, the vehicles follow a preplanned path to cross the intersection. This path is simplified to match the tiles used in the intersection. The car body is represented by a circle to simplify the simulation and collision checking. The circle has a diameter of 2 meters while the lanes are 3 meters wide. We assume a 15% rate for both right and left turns, 70% of the cars are thus trying to move straight across the intersection. Each starting lane has a corresponding end lane. Lane swaps are prohibited.



(a) Intersection Layout **(b) STARC protocol delay without platoons** **(c) STARC and Platoon Extension Compared to Traffic Lights**

Figure 3.2. ***(a)*** **The simulated intersection has three ways per direction. The arrows represent the movement restrictions with the corresponding tiles in the 6×6 grid.** ***(b)*** **The STARC protocol simulations without platoons indicate average delays of less than 20 seconds below 1000 vehicles per hour and congestion afterward.** ***(c)*** **While STARC without platoons shows less delay than traffic lights only below 1000 vehicles per hour, the version with platoons dominates even in higher traffic levels.**

Implementation. We use the Synchrotron implementation for TelosB sensor nodes [75]. The TelosB features an MSP430 chipset by Texas Instruments with 10 kB RAM and a 4 MHz CPU [129]. With their 250 kbps IEEE 802.15.4 radio, the nodes act as transmitters for the STARC protocol. They run Contiki OS, an open-source operating system for the Internet of Things [130]. We use the simulator Cooja to emulate the nodes' code execution and simulate their radio messages using a Multi-path Ray-tracing radio medium for an almost realistic radio simulation.

3.4.1 Radio Failures

Wireless communication is prone to error and unreliable by nature: interference from other technologies, multi-path reflection, and antenna orientation can significantly affect the quality of communication. We evaluate STARC's resiliency against failure and its safety property when some cars are unable to communicate with the rest of the network.

Scenario. We inject random failures blocking communication for certain nodes in the network. During a round, at each slot, each node has a probability of failing. Upon failure, the node is unable to communicate: it does neither receive nor transmit any message until the end of the round. All failed nodes recover their communication capabilities once the next round starts. We exclude the leader from failure injection: the leader is always able to communicate. We fix the traffic to 1000 incoming cars per hour.

Results. Table 3.2 presents the agreement success rate and the number of car collisions for specific failure rates. Under lower failure rates of up to 0.01%, most rounds are successful, and cars agree on a crossing schedule. With a failure rate of 0.1%, each car has a probability of 0.1% to stop communicating every 6 ms, and the success rate drops to 63.7%. This means that 6 out of 10 rounds led to an agreement on which cars should cross the intersection. In 4

Table 3.1. Evaluation parameters

	Parameter	Value
Intersection	Number of lanes	12
	Tile grid	6×6
	Turn rate (left & right)	15 % each
	Lane width	3 m
	Maximum speed	$50\,\mathrm{km\,h^{-1}}$
Vehicle	Acceleration	$2\,\mathrm{m\,s^{-2}}$
	Deceleration	$4\,\mathrm{m\,s^{-2}}$
	Vehicle diameter	2 m
STARC	Round Interval	2 s
	Slot length	6 ms
	Slot number	200
	Max. network size	16
	Join / leave / rejoin slots	4 / 16 / 1

Table 3.2. Commit success and collisions in the presence of radio failures. While failures affect the commit success rates, no collisions occurred in the simulations.

Failure Rate	Commit Success Rate	Car collisions
0 %	99.8%	0
0.001%	99.4%	0
0.01 %	96.1%	0
0.1 %	63.7%	0

out of 10 rounds, the network did not commit on a new schedule, but no car disagreed and chose its personal, unsafe schedule: all cars agreed not to cross.

For all failure rates, no collision between cars happened. STARC is safe since it prefers to block crossings to prevent conflicting paths and possible collisions under failures.

3.4.2 Delay Induced by Crossing

Scenario. We evaluate STARC's efficiency with regards to varying traffic load. Here, we define efficiency as the delay experienced by a user from the time the vehicle reached the intersection up to the point the vehicle leaves the intersection, and the STARC network. We vary the traffic load from very few cars up to 1200 vehicles per hour, which corresponds to a medium city intersection in the morning hours with one car every 3 seconds.

Results. Figure 3.2b depicts the average delay experienced at the intersection running STARC. While the average delay is below 20 seconds at first, the delay spurts upwards at more than 1100 vehicles per hour. However, the lower bound on the confidence interval indicates that some vehicles still experience less delay.

Figure 3.3 breaks down vehicle delays for 1100 cars per hour for different directions. With 70% of the vehicles driving straight over the intersection, the straight-driving vehicles spent most of the time (38.2 s) waiting in the

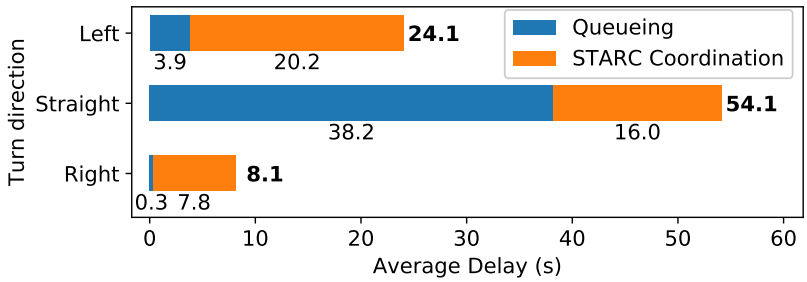


Figure 3.3. A delay composition for each turn direction in STARC simulation runs with 1100 vehicles per hour and no platoons. Joining and leaving the network, as well as waiting for the reservation, are part of the coordination. The waiting times correspond to different path lengths. Driving times are excluded. 70% of the cars continue straight.

queue. This time is lower for vehicles that turn left (3.9 s) or right (0.3 s). The waiting times for acceptance corresponds to the number of tiles needed for the reservation: right-turning vehicles have to wait for the least (6.2 s), while left-turning vehicles have to wait for the longest (18.9 s). The straight-driving vehicles experience intermediate waiting times (14.6 s). The time to leave the network is nearly equal across all directions (1.3 s to 1.5 s).

3.4.3 Traffic Lights Comparison

Scenario. The scenario is modified to support traffic lights, but its basics stay the same. When passing over a traffic light intersection, the cars only have two states: queueing and moving. Cars have to queue and wait for a green light before they may cross the intersection. The traffic lights use a simple all-lane-model: All lanes of one direction are allowed to drive simultaneously [79]. Each direction is scheduled once a minute, resulting in 15 seconds per direction. Out of those 15 seconds, a green light is shown for 9 seconds. A yellow and red light show for 3 seconds each, assuring a safe passing of potential left-turning vehicles. In the scenario with traffic lights, only the time for queueing specifies the delay. We also evaluate the platoon extension of STARC, as described in Section 3.3.3. We limit the platoons to a maximum size of 25 vehicles and only allow joining the platoon before it starts crossing. The platoon head coordinates and reserves the path for the full platoon.

Results. Figure 3.2c presents the average delay of traffic lights and the STARC protocol without and with platooning enabled. The platoon size limit of 25 roughly equals the maximum amount of cars that could cross straight during a single green phase. The recorded delay for our protocol without platoons increases slowly, and at more than around 900 vehicles per hour, skyrockets. At around 1000 vehicles per hour, the delay without platoons already surpasses the delay of traffic lights. This limit is not surprising, considering that the protocol without platoons allows only a single vehicle per lane to move. The version with platoons handles those values well and does not only compete with the traffic lights but also further decreases the delay in low traffic settings

with less than 1000 vehicle per hour compared to STARC without platoons. All mechanisms can not handle arbitrary high traffic volumes. Just as a longer green phase for the traffic lights, we expect that a higher limit for the platoon size could reduce the experienced delay in higher traffic volumes (without increasing the delay for lower traffic volumes).

3.5 Conclusion

Crossroads are the bottlenecks of road networks. Intersections often require infrastructure, such as traffic lights, to enable coordinated crossing and ensure the safety of all road users. Connected traffic is a building block of Intelligent Transportation Systems: if vehicles and other road users can communicate, they can coordinate how to efficiently and safely cross intersections. While centralized coordination mechanisms could allow optimal crossings, they require the presence of a central infrastructure, e.g., LTE or 5G coverage.

We introduce STARC, a decentralized reservation-based protocol for safe intersection crossing. STARC uses cheap, low-power wireless radios and builds on top of IEEE 802.15.4 to enable energy-efficient vehicle-to-vehicle communication in the absence of cellular coverage. With STARC, road users can instantly create or join a low-power local network to coordinate at intersections. We show through simulations that STARC is safe, reduces average waiting times by up to 50% compared to traffic lights for volumes lower than 1000 vehicles per hour, and can efficiently serve traffic loads over 1000 vehicles per hour with the support of platoons.

Our future work will investigate priority strategies supporting high priority vehicles such as ambulances.

4

Dimmer: Self-Adaptive Network Floods with Reinforcement Learning

---

V. Poirot, O. Landsiedel

Under submission.

PAPER C

Abstract

Efficient, highly-reliable communication primitives are a cornerstone of low-power wireless networks. Through network-wide flooding, Glossy and higher-level primitives such as Low-power Wireless Bus (LWB) are able to provide message delivery with high reliability and low-latency. However, these protocols suffer from two limitations: (1) they are invariant to their environment dynamics and deal with interference through over-provisioning, and (2) their one-fits-all, cradle-to-grave designs lead to wasted energy in dense parts of deployments and interference-free periods.

We argue that low-power wireless networking should *adapt* to the wireless medium to meet a target performance, even under varying conditions, while still ensuring energy efficiency. We propose *Dimmer* as a self-adaptive, all-to-all communication primitive. Dimmer builds on top of LWB and uses Reinforcement Learning to tune the flooding parameters to match the current properties of the medium. By learning how to behave from unlabeled traces, Dimmer adapts to different interference types and patterns, and is even able to tackle previously unseen interference. Through Dimmer, we share insights on how to efficiently design AI-based systems for constrained devices, and evaluate our protocol on two deployments of 18 and 48 resource-constrained sensor nodes (4 MHz CPU, 10 kB RAM), showing it improves reliability under WiFi interference and IEEE 802.15.4 jamming.

4.1 Introduction

Context. Reliable, energy-efficient communication is a cornerstone of low-power wireless networks. Network-wide flooding, with its flagship Glossy [23], has been established as an important building block to compose such efficient protocols [27, 28, 131]. Building on top of Glossy floods, Low-power Wireless Bus (LWB) provides low-latency and highly reliable message delivery, and can be seen as the de-facto protocol for many-to-many communication in low-power wireless networks [27]. By design, LWB is topology-independent, and with its central scheduler, LWB adapts to changes in traffic. However, Glossy, and thus LWB, suffer from two limitiations: (1) both protocols are invariant to dynamics in their environment, and deal with interference through over-provisioning [32]; and (2) in the absence of interference, their one-fits-all approach leads to wasted resources, especially in dense regions of deployments [132, 133].

Firstly, the performance of Glossy and LWB degrades as interference arises. To tackle this, improvements over Glossy use channel-hopping and tend to over-provision resources, e.g., by increasing the number of retransmissions within a flood [32, 134]. This over-provisioning is, however, static for a deployment's lifetime, from its design-cradle to its grave, and thus independent of the actual experienced interference. This inability to adapt to the environment leads to wasted energy in the case of transient wireless perturbations. Secondly, all devices participate in the message propagation. Thus, in dense regions, many forwarders concurrently transmit, which can impede the reliability of the flood as well as unnecessarily deplete energy. To counter this, XFCS and Sleeping Beauty use additional communication to select active forwarders, putting superfluous nodes to sleep [132, 133], while Zhang et al. use multi-armed bandits to learn the optimal retransmission parameter in Glossy at runtime, assuming no interference is present [93].

Goal. Our goal is a paradigm where a low-power wireless stack adapts itself to the wireless channel and deployment topology to meet a target performance, while ensuring energy efficiency and efficient utilization of the wireless medium, as depicted in Fig. 4.1.

Challenges. Building a self-adaptive wireless stack can be decomposed into two distinct sets of challenges: (1) adapting to interference, and (2) adapting to the topology, to achieve energy-efficiency in the interference-free case. Interference comes in many shapes, e.g., bursts or slow channel-fading, duration, and patterns [31, 135]. Designing a general solution resilient to all interference is non-trivial; a protocol designer must deal with limited available information, and rely on its expert knowledge, e.g., to estimate when the interfering episode has passed. This often leads to handcrafted and carefully fine-tuned decision rules, that sometimes need to be re-tuned for each new deployment and application. As example, PID controllers, the go-to approach in closed-loop control, must be tuned, either through experimentation or complex numerical methods, prior to their deployments, and possibly re-tuned to fit each deployment [136–138]. Furthermore, our self-adaptive protocol must react quickly to sudden interference surges. In contrast, distributed and autonomous solutions are notoriously slow to adapt, and are prone to instability [93]. To solve these challenges, we focus on quick, centrally-controlled adjustments of our communication substrate once interference is detected. We implement deep

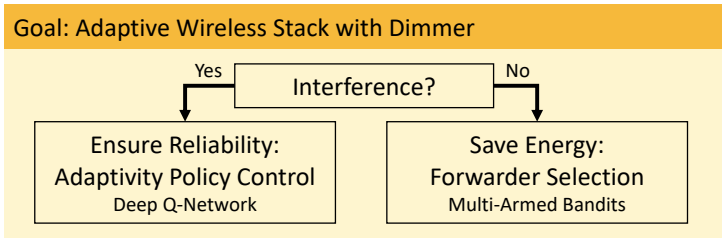


Figure 4.1. Adaptive wireless stack. Smart communication protocols must adapt to changes to the wireless medium, and save energy under good conditions. With Dimmer, we use the advances in AI to bring adaptivity to low-power wireless networks.

reinforcement learning methods using neural networks to obtain a self-adaptive stack able to learn the optimal retransmission parameter over unlabelled traces, in the absence of any human supervision.

During interference-free periods, a self-adaptive flooding protocol must ensure energy-efficiency. Forwarder selection has been shown as an effective method to save energy in dense deployments. However, Glossy does not provide neighbor information, and XFCS and Sleeping Beauty rely on additional control transmissions to select forwarders [132,133]. We employ a distributed forwarder selection scheme using multi-armed bandits to find superfluous forwarders at runtime, at no additional communication costs, by extending the approach introduced by Zhang et al. [93].

System Challenges. Building on neural networks and deep reinforcement learning brings its own, specific set of challenges: (1) We have to capture the dynamics of the wireless medium in a neural network so that it can learn how to quickly and efficiently adapt to the network dynamics. (2) It is practically impossible to obtain a dataset of transmissions with optimally selected parameters to train on, as used with supervised methods. Thus, we must build an unlabelled simulation environment in which a reinforcement-learning agent can be trained. (3) Low-power wireless systems are resource-constrained, in the order of several MHz and tens of kB of RAM, and demand for space-efficient neural networks so that we can deploy them on the hardware. (4) Finally, multi-agent RL systems where each node execute its local multi-armed bandit (MAB) instance are prone to unstability. A distributed MAB approach must ensure that the learning phase does not lead to oscillatory behavior and degraded performance.

Approach. We introduce *Dimmer*, a self-adaptive, all-to-all communication primitive based on LWB [27], the de-facto protocol for data collection and dissemination in IEEE 802.15.4 low-power wireless networks. Combined with LWB's central scheduling, Dimmer introduces two novel elements: (1) a centrally-executed deep Q-network that adapts the retransmission parameter to tackle interference, and (2) a distributed forwarder selection scheme using multi-armed bandits at runtime, to save energy in the interference-free case. Dimmer does not require extra-communication: the feedback loop is closed by collecting performance metrics alongside data packets, and disseminating update decisions along with LWB schedules. Upon detecting packet losses, our

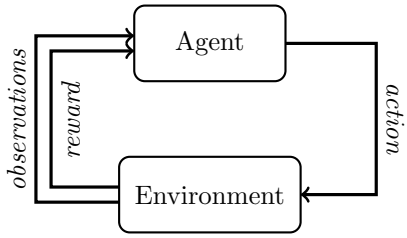


Figure 4.2. Reinforcement Learning (RL): A learning agent interacts with its environment through actions. The environment returns observations and a reward.

central neural network globally increases the number of retransmission within a flood. The deep Q-network is then able to detect once interference has passed and converges back to its optimal parameter. During interference-free periods, Dimmer nodes locally learn if they are superfluous during flood propagation, or must act as active forwarder, using a multi-armed bandit approach. While the approach of Zhang et al. focuses on optimizing a single-source flood, Dimmer extends the optimization to multi-source communication.

Contributions. This paper contributes the following:

- We present Dimmer, an RL-enabled self-adaptive communication primitive featuring a deep Q-network and multi-armed bandits;
- We highlight how we map Dimmer to a solvable RL problem, and how we design a constrained embedded deep Q-network fitting to a platform featuring a 4 MHz CPU and 10 kB of RAM;
- We introduce a trace-based environment that Dimmer uses to learn interference-coping strategies in a matter of minutes rather than days;
- We provide an open-source implementation for TelosB motes and evaluate our solution on two testbeds comprising 18 and 48 nodes, showing it is able to operate on new topologies and adapt against unseen interference without retraining its DQN.

The remainder of the paper is structured as follows: we introduce reinforcement learning and LWB in §4.2. We give an overview of Dimmer in §4.3, explain how we solve typical RL challenges in §4.4, and go deep into Dimmer's design in §4.5. In §4.6, we evaluate Dimmer in depth on testbed deployments. Finally, we discuss related work in §4.7 and conclude our work in §4.8.

4.2 Background

We introduce the basics of reinforcement learning, Q-learning, and multi-armed bandits in §4.2.1. We then present the LWB primitive in §4.2.2.

4.2.1 Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of machine learning [54]. It targets *sequential decision making*; in an RL problem, an agent learns how to achieve

a complex task by taking consecutive actions. RL differs from both supervised and unsupervised techniques: in supervised learning, a dataset of input-output tuples is available, and the goal is to find a generalized function mapping them. In unsupervised learning, an unlabeled dataset is available, and the goal is to find if a hidden structure exists within. In contrast, in RL, not only is the data often unlabeled, but the algorithm must discover the sequential steps of transitions that bring its final, desired state.

Trial and error. To discover the optimal strategy, an agent must interact with its environment and learn from its mistakes. Fig. 4.2 illustrates how a RL agent behaves during learning. At first, the agent *observes* its environment. Based on the observed state of its surroundings, the agent makes a decision and *acts* on it. In doing so, the agent alters the internal state of the environment. The agent can, therefore, observe the effects of its action, and can use a *reward* signal to evaluate the usefulness of its actions.

By exploring the environment and trying random actions, the agent can accumulate experiences and build an internal model of how its actions affect the environment. By exploiting its reward and with enough accumulated knowledge, the agent can construct a sequence of actions that solves the complex task at hand.

Markov Decision Processes. An RL problem is said to be solvable if the environment can be represented as a Markov Decision Process (MDP) [55]. MDPs extend Markov chains: in Markov chains, the transition from a state to a new state is represented as a probabilistic distribution. In an MDP, a decision additionally affects the transition. As example, in the case of a network-wide flood such as Glossy [23], where each node repeats multiple times the packet in order to increase the probability of reception, the transition is affected both by an a priori random wireless interference, as well as a controllable parameter, the retransmission count. More formally, am MDP is represented by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ the set of possible actions, $\mathcal{P}$ the transition probability function and $\mathcal{R}$ a reward function.

Q-Learning. Finding the best sequence of actions requires an agent to estimate future rewards. The agent seeks to maximize the cumulative reward $R_t \triangleq \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$, where r_τ is the reward obtained when transitioning at time τ, and $\gamma \in [0, 1)$ a constant called the discount factor. A small discount factor will force the agent to maximize short-term, immediate rewards, while a high discount factor will allow the agent to maximize long-term expected rewards.

Q-learning is one of the most popular ways to solve RL problems [54]. In Q-learning, an agent learns an action-value function $Q(s, a)$. This function represents the expected cumulative reward the agent should get when starting in state s, using action a such as:

$$Q(s, a) \triangleq \mathbb{E}\big[R_t \mid s_t = s, a_t = a\big] \tag{4.1}$$

In simple terms, the Q-function evaluates how valuable it is to choose action a in state s, in terms of expected reward.

If the environment can be modeled as an MDP, then we can find an optimal function $Q^*(s, a)$ that follows Bellman's principle of optimality [56]:

$$Q^*(s, a) = \mathbb{E}\big[r_t + \gamma \max_{a'} Q^*(s', a') \mid s_t = s, a_t = a\big] \tag{4.2}$$

where r_t is the immediate reward received, γ the discount factor presented above, and s' the state achieved after the state s. By iteratively trying actions and receiving rewards, we can update a Q-function that ultimately converges to the optimal $Q^*(s, a)$.

Deep Q-learning. While Q-learning algorithms have historically used a tabular approach in estimating the Q-function [54], deep neural networks have been recently established as Q-function approximators [57]. Deep Q-networks (DQN) have been successfully used to solve problems outside of their original application: datacenter cooling [58], wireless modulation [59], CSMA/CA optimization [60], etc. One advantage of DQN over tabular approaches is the ability to solve problems with continuous states and the generalization property of neural networks.

Multi-armed bandits. In the Multi-Armed Bandits (MAB) problem, a gambler is facing K machine slots in a casino, each machine giving an a priori unknown, stochastic reward upon pulling its arm [139]. The goal of the gambler is to maximize its returns in a minimal number of steps, by carefully controlling its exploration and exploitation trade-off. In the extended adversarial MAB setting, an adversary is able to impact the reward system associated to each arm [140]. In wireless systems, changing conditions of the medium can be represented as an adversarial setting. The gambler thus cannot rely on past experiences only, and must continuously explore. Exp3 is an online algorithm solving the exploration-exploitation trade-off in adversarial MAB [141]. Exp3 associates an exponential weight with each arm, thus leading to quick adaptation to adversarial changes in the environment.

4.2.2 Low-power Wireless Bus

In low-power wireless networks, quickly flooding a message to the entire network has established itself has a simple and efficient method to provide communication, in contrast to slow, expensive routing solutions [142].

Glossy. Glossy is one of the pioneer works in synchronous transmissions [23]. Through tight synchronization (< 0.5 μs) and by sending identical data, Glossy provides network-wide broadcasts (or floods) with high reliability ($> 99.9\%$) and low power-consumption. Within a Glossy flood, a packet is retransmitted multiple times, typically 3, and nodes alternate between transmission and reception to keep synchronization and reduce the number of concurrent transmissions.

LWB. Low-power Wireless Bus (LWB) is a flexible communication protocol tailored to wireless sensor networks [27]. Its most prominent feature is its ability to support many different traffic patterns. LWB uses Glossy floods as communication primitive, effectively turning a multi-hop network with mobile nodes into a logical bus, in which any node can potentially receive any packet, without the need for expensive routing.

LWB is a centralized solution; a host node computes a schedule that satisfies flows requested by (message)-source nodes, and controls the periodicity of communication to save energy. As such, LWB is a versatile solution for low-power communication. The protocol is also used as baseline for the 2019 EWSN dependability competition [143].

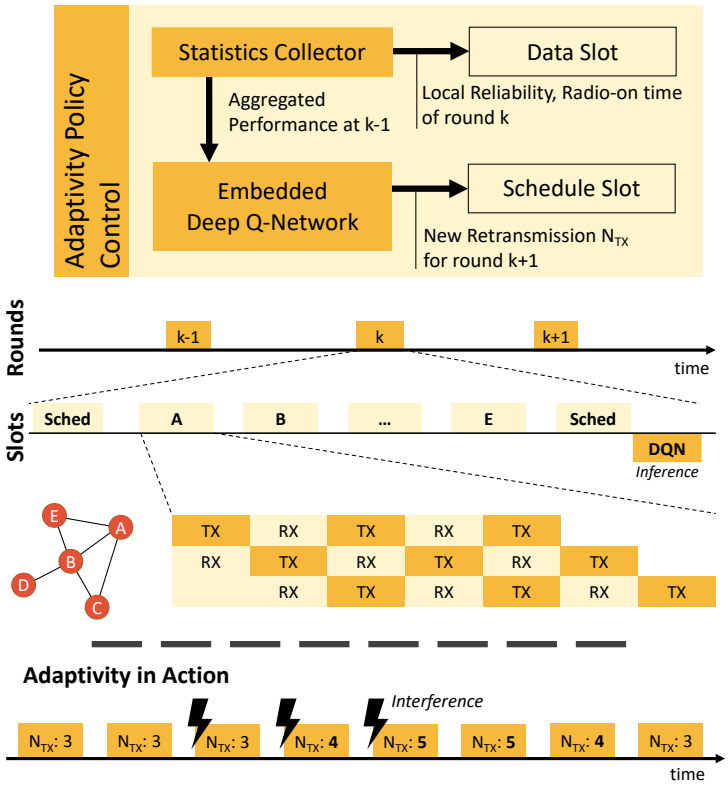


Figure 4.3. Adaptivity Policy Control in Dimmer. At the end of the round, the coordinator executes its deep Q-network based on previous network performance, and disseminates a new retransmission parameter N_{TX} along with the next schedule. Source nodes append to their payload their performance during their data slots, thus closing the feedback loop.

4.3 An Overview of Dimmer

We introduce Dimmer, an adaptive all-to-all communication primitive for resource-constrained devices. Dimmer builds on top of LWB [27], and integrates two novel components alongside LWB's central scheduler: (1) a central adaptivity control scheme, used to quickly adapt to interference surges through an embedded deep Q-network, and (2) a distributed forwarder scheme, allowing Dimmer to discriminate superfluous transmissions at runtime and save energy through the use of multi-armed bandits.

Dimmer in a nutshell. Like LWB, Dimmer operates in communication rounds, see Fig. 4.3. A round is composed of a schedule slot, sent by a central coordinator, followed by a series of attributed data slots. During each data slot, the source node appends to its payload four bytes of meta-data on performance, such as reliability and radio-on time. These metrics, collected by the coordinator and participating nodes, compose an up-to-date global snapshot of the network performance. The communication round ends with a second schedule, used

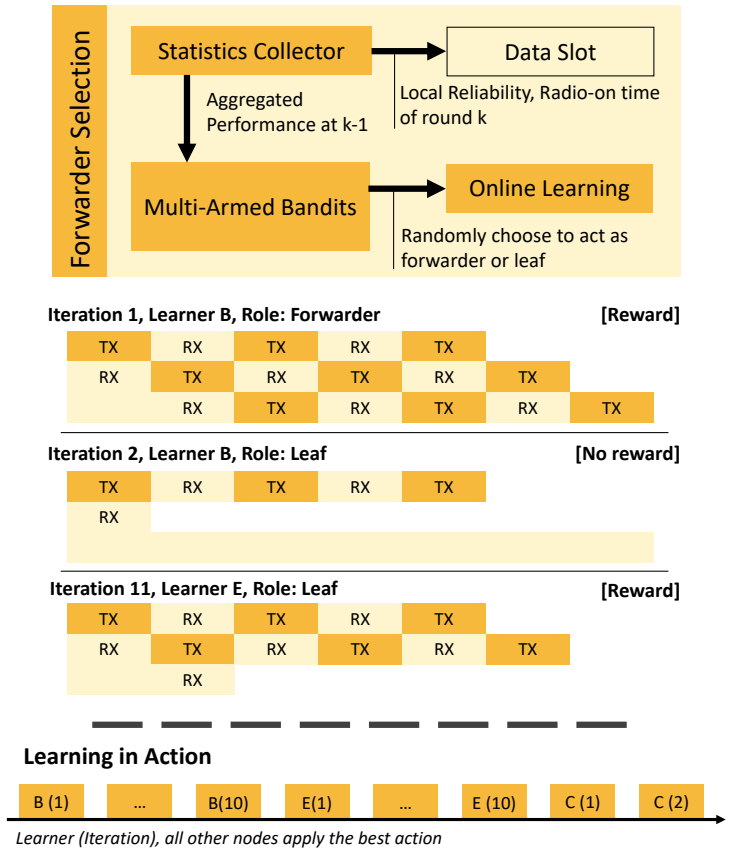


Figure 4.4. Forwarder Selection in Dimmer. In the interference-free case, nodes take turn learning whether they should act as forwarder to help the flood dissemination, or leaf to save energy.

to coordinate the next round. Once the communication round has ended, the central coordinator executes its embedded DQN to adapt to any changes to the medium. If the medium is free of interference, the nodes instead execute their distributed forwarder selection scheme. In addition to dynamically adjusting the retransmission count, Dimmer tackles interference with frequency diversity. Each data slot is attributed a channel to use from a globally known hopping sequence, as used in Crystal [28].

AI-augmented wireless. With Dimmer, we argue that recent advances in AI, and specifically reinforcement learning, can be used to design self-adaptive primitives. Traditional solutions, such as handcrafted rule-based systems or PID controllers, the go-to approach in closed-loop control, require expert knowledge during the design phase, e.g., an understanding of possible disturbances to the medium, how to measure them, and mitigate their effects [137, 138]. Further, once the solution has been tuned, there is no guarantee that it will provide similar performance on a new deployment or on a different hardware. With machine learning, deploying to a different topology or changing the hardware simply equates collecting new traces and retraining the neural network, and

can be done as an automatic step during deployment. Moreover, reinforcement learning does not require a dataset labeled with the optimal retransmission parameters, as the RL agent will use trial-and-error to build a model of its environment and learn how to act optimally. This means that using Dimmer does not require any prior expert knowledge. In addition, we show in §4.6 that deploying Dimmer in a new environment does not always equate retraining the DQN, as we demonstrate by operating Dimmer on a 48-node deployment against WiFi jamming, while the DQN was trained on traces collected on a 18-node testbed predominantly featuring 802.15.4 jamming.

Embedded central adaptivity control. Dimmer incorporates a central adaptivity control scheme using an embedded deep Q-network (Fig. 4.3). Because interference quickly rises, distributed solutions are too slow to react, or might even cause instability [93]. As soon as a communication round has ended, the central coordinator executes its DQN, using the worst-nodes performance as input, and the current retransmission parameter. During the next communication round, the coordinator disseminates the new retransmission parameter along the round schedule. With Dimmer, we do not assume the presence of a powerful edge-device to offload computation to; swarms of drones cannot maintain constant communication and rely on distant computing to react to interference. Instead, we choose to embed our intelligence onto the resource-contrained node executing Dimmer. Thus, we quantize our DQN to fit within 10 kB of RAM and to perform on a 4 MHz CPU.

Distributed Forwarder selection. We model the forwarder selection problem of Dimmer as an adversarial multi-armed bandit setup and solve it using Exp3 [141], by extending the approach introduced by Zhang et al. [93]. In their work, they set to optimize the retransmission count of each node during a single-source, Glossy flood. In contrast, we tackle the problem of optimizing multiple, multi-source floods, and model the decision as a binary choice of acting as forwarder.

Whenever the medium is free of interference and performance is good and stable, Dimmer nodes execute the forwarder selection scheme in an iterative manner, where at most one node learns at a time, see Fig. 4.4. Exp3 computes the probability of each node to act as forwarder. At the beginning of a round, the learning node draws its decision (to act as a forwarder or not) from Exp3's probability. All other nodes keep their previously learned behavior. The learning node then collects the feedback from Dimmer, and updates its probability based on whether any flood showed degraded performance. After several iterations, we obtain a configuration where superfluous nodes learn to not act as forwarder, thus saving energy by turning off their radios earlier.

Design outline. In §4.4, we introduce Dimmer in detail and discuss how it solves key challenges of designing wireless primitives using RL. Next, in §4.5, we describe the system design, i.e., how the Dimmer primitive operates on constrained hardware: §4.5.1 presents our architecture, §4.5.2 our central adaptivity control using the DQN, and §4.5.3 presents our forwarder selection scheme using multi-armed bandits.

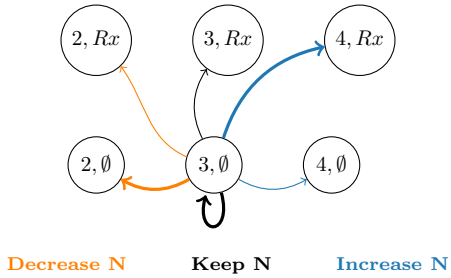


Figure 4.5. Glossy as a Markov Decision Process. For each retransmission factor N, (N, Rx) represents the state where last transmissions were successful, while $(N, \emptyset)$ represents an interfered medium. We depict transitions from $(3, \emptyset)$ only. In each state, an agent can either decrease, increase or keep the same N value. The thickness of the arrow depicts the transition likelihood. Increasing N here leads to an increased chance of successful floods.

4.4 Designing a RL-based protocol

Challenges. To model Dimmer's central adaptivity control as a RL problem, we must solve the following problems:

C1. Defining a solvable RL problem: we represent Glossy and Dimmer as MDP.

C2. Selecting actions and features: we show that it is sufficient to use a limited set of features and only three actions.

C3. The reward leads to a correct behavior: we define a reward function with a given reliability goal and minimization of the energy consumption.

C4. The neural network fits on a resource-constrained device: we show that a quantized neural-network with one hidden layer of 30 neurons is sufficient.

C5. The neural network is (efficiently) trainable: we devise a trace-based training environment.

Additionally, to solve the forwarder selection problem, we must ensure that:

C6. The forwarder selection does not cause instability: we serialize Exp3's execution.

We discuss how we solve challenges C1 to C4 with Dimmer in §4.4.1, deal with C5 in §4.4.2, and tackle C6 in §4.4.3.

4.4.1 Resource-constrained RL Problem

C1. Dimmer as solvable RL problem. We define the following optimization problem: Dimmer must find the optimal N parameter, i.e., the number of packet retransmission within a Glossy flood, that maximizes the network reliability while minimizing the energy consumption at any time. Concretely, this means: (1) Under normal conditions, Dimmer should achieve a very good

Table 4.1. Input vector of Dimmer's DQN. Parentheses denote the number of elements used by Dimmer during evaluation.

Input	Number of rows (31)	Normalization
Radio-on time	K (10)	[0, 20ms] → [-1,1]
Reliability	K (10)	[50, 100%] → [-1,1]
N parameter	$N_{max}+1$ (9)	One-hot encoding
History	M (2)	-1 if losses, otherwise 1

reliability ($> 99.9\%$) without wasting energy, i.e., by avoiding unnecessary retransmissions. (2) Dimmer must adapt once it detects interference, i.e., once it detects a drop in reliability. We allow Dimmer to lose a few packets while adapting to changes in the channel conditions. In §4.2.1, we state that an RL problem is solvable if it can be represented as an MDP. In the following, we introduce an MDP representation of Glossy and, by extension, Dimmer.

We define two states for each possible N value, (N, Rx) and $(N, \emptyset)$, see Fig. 4.5 for an extract of our MDP. Each state represents the current retransmission factor N, tupled with the outcome of the current flood: Rx if transmissions are successful, otherwise $\emptyset$. In each state, an agent can either decrease, increase or maintain N. The external interference is modeled by an unknown transition probability $\mathcal{P}$ (see §4.2.1). The goal of the RL algorithm is to recognize its current state, and learn how to best adapt N to reach the most beneficial state.

C2.1. Defining a small action space. To foster quick learning and ensure a small neural network, it is essential to keep the action space small. While it is possible to define an action for each retransmission value (e.g., an action for N=3, one for N=4, etc.), this approach leads to an action space too large to execute on embedded hardware. Further, we argue that, in our experience, such learning easily overfits our environment specifics and is unlikely to generalize to unseen interference patterns. Instead, we define three actions: *Decrease*, *Keep*, and *Increase* N. Our approach leads to a generalised behavior: a system that learned to increase under interference is able to deal with any reliability change, irrespective of the current N value. A drawback is that the system is limited to step-wise increase, e.g., going from N =1 to 4 takes three steps.

C2.2. Selecting meaningful features. In order to close the loop, we next define the features – in our case network metrics – we collect as feedback for the actions of the RL-agent. Each node shares two statistics over the network: their *radio-on time*, as a proxy for energy consumption, and their recent *reliability*. Since statistics are piggybacked along with data packets, they do not require additional floods.

Neural networks suffer from their rigid structure: their input vectors must remain constant in size. If a neural network is trained using one input per sensor node, the neural network needs to be entirely retrained if a node is added or removed. To counter this limitation, our DQN requires input from a subset of nodes only. Thus, Dimmer supports a wide range of deployments of varying size without retraining. We select the K nodes with decreasingly worse

reliability, to correctly represent the suffered packet losses. If no feedback is received from a given node, Dimmer assumes that node has 0% reliability. In addition to these K network statistics, Dimmer uses the current retransmission parameter N as input. N is represented as one-hot encoding, i.e., we use a vector of size $N_{max}+1$, where the row N is set to one, and all other rows set to zero. Finally, we introduce additional, historical reliability features. Historical features allow Dimmer to avoid oscillatory behavior under interference. Table 4.1 summarizes the input vector.

We normalize each input feature between -1 and 1. We deem any reliability lower than 50% as unacceptable, and and represent it as -1. For historical features, we represent the previous round as -1 if at least one packet was lost, and 1 if all packets were received by all nodes. Thus, we obtain an input vector with 31 elements. This enables a small neural network while being able to support a wide range of deployments with varying size.

C3. Reward engineering. In RL, the goal of an agent is to maximize a reward function, which mathematically represents the problem itself. Following our goal stated in Fig. 4.1, we design Dimmer's reward to obtain the following behavior: reliability should be maintained at all costs, but energy should be saved under good conditions. We model our goal with the following reward function:

$$r_t \triangleq \begin{cases} 1 - C * N/N_{max}, & \text{if no losses} \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

where N/N_{max} is the normalized retransmission factor and $C = \frac{3}{10}$ a constant. The value of C impacts the reliability-energy trade-off: For high values, the RL agent will converge to a lower N value under good condition; for low C values, the agent will tend to keep many retransmissions to avoid any losses.

C4. Selecting neurons. In C3 we define the input and output spaces used. For the feature set we eventually choose, c.f. §4.6, Dimmer requires 31 input features and outputs an expected reward for 3 possible actions. Knowing that our DQN is to be executed on resource-constrained devices (4 MHz, 10 KB RAM, no floating point), we go for the smallest, functioning architecture. We use two fully connected layers; a hidden layer of 30 neurons with rectified linear (ReLU) activation, plus three neurons for the output layer. We motivate this choice by the light computation cost of ReLU, as well as their effectiveness in the original DQN [57]. We explain in §4.5.2 how we quantize our DQN to fit on a resource-constrained device.

4.4.2 Learning From Traces

Like all machine learning techniques, reinforcement learning requires access to data. However, unlike other ML approaches, RL learns through interaction: an accurate simulator or access to a real deployment is necessary for training, rather than a dataset of labeled instances. In this section, we discuss how we build an accurate simulator based on traces collected from a real deployment, on which we are able to train Dimmer.

C5.1. Collecting meaningful traces for training. RL agents improve their Q-function by infinitely trying state-action pairs. While RL relies on observing the immediate effect of an action, we argue that we can collect traces

that accurately represent the impact of an action under similar wireless medium dynamics. In §4.4.1, we explain that Dimmer can be modeled as an MDP, meaning that the Markov property holds [55]. Thus, building a simulator from a collection of traces is possible if we ensure that the traces correctly model the dynamics of the wireless medium.

We define a datapoint in our traces as the combined results of all slots during one round. Because a round is composed of several floods, our trace represents fast interference, e.g., a transient disturbance in the order of tens of milliseconds, as a single datapoint with low reliability. In contrast, our environment represents slow-moving interference, e.g., numerous bursts spaced over a second, as a decrease in reliability over multiple datapoints. A correct training environment must represent this slow dynamism: we iterate over all possible N values sequentially during the trace collection, and ensure that during training, consecutive actions draw timely-coupled datapoints, i.e., data that are close to each other in the trace. Thus, we guarantee that slow-moving interference is correctly modeled in our training environment.

Representative traces. We ensure that our traces are representative of the environment by collecting them over multiple days, for different times of the day and frequencies. Representative traces must correctly model the distribution of the real-world scenario: interfered scenarios should not be overly represented in the traces if Dimmer is to be executed in normal deployments. Our traces are collected both during days and nights, with colocated WiFi traffic from typical office environments and student lab rooms. Further, some traces are collected with IEEE 802.15.4 jamming active, where the interference pattern is changed several times throughout the trace, to represent a larger sample of possible disturbances.

C5.2. Offline learning. Our targeted platform, a TelosB featuring a 4 MHz CPU and 10 kB of RAM, is unfit to support the training of our DQN. Therefore, we train our neural network offline, and embed the result of the learning on the resource-constrained device for its inference step. We train our DQN for 200 000 iterations with an epsilon-greedy selection scheme. The selection probability in annealed from 100% to 1% linearly over the length of 100 000 steps, and fixed to a random action probability of 1% afterwards. We select a discount factor (see §4.2.1) of 0.7.

4.4.3 Distributed Forwarder Selection

The forwarder selection problem, i.e., selecting a subset of nodes that must transmit during a flood to help the dissemination, is computationally expensive for a centrally learned scheme. For a network with n nodes, there is 2^n possible configuration of forwarders and leaf nodes. Assuming one wants to train a RL agent to solve this task, we estimate that building a training environment requires well over 20 days of data per trace, where multiple traces are required to avoid overfitting. Instead, in Dimmer, we learn how to act in an online fashion, i.e., at runtime, and divide the problem into a localized decision, i.e., each node decides if it should act as forwarder. We get inspiration from the approach of Zhang et al. and use Exp3 to design our solution [93].

C6. Serialized Exp3. As demonstrated by Zhang et al., using Exp3 to solve the adversarial multi-armed bandits is not guaranteed to converge to a

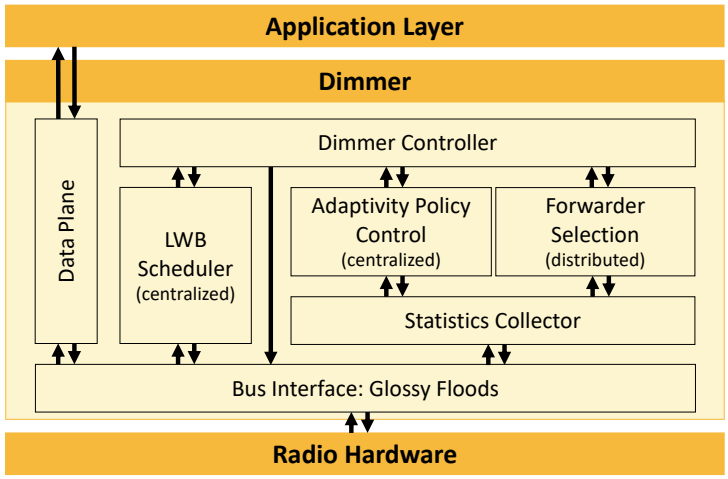


Figure 4.6. Dimmer's architecture. Dimmer combines an adaptivity policy control unit featuring a DQN to adapt to interference, and a forwarder selection scheme built on multi-armed bandits to save energy.

stable state. While we cannot guarantee convergence ourselves, we introduce three techniques to avoid oscillatory states that can degrade the network performance. Firstly, we serialize the learning execution. At any round, at most one node can randomly choose an action and learn from the feedback. Further, a node is given 20 consecutive rounds to try and evaluate different strategies and to converge to a decision. Secondly, whenever a node chooses to not act as forwarder and degrades performance in doing so, we further pushes the node to act as a forwarder later, by giving an additional reward for the 'forwarder' decision. Finally, we use a pseudo-random order for the serialized execution. This allows us to avoid network-breaking configurations in deployments where geographically close nodes would otherwise be potentially always in the same order.

4.5 System Design: Dimmer, an All-to-All adaptive primitive

After discussing the RL-oriented design choices of Dimmer, we now introduce the integration of Dimmer into the LWB protocol stack. We dive into the architecture of Dimmer in §4.5.1, its central adaptivity control in §4.5.2, and its forwarder selection scheme in §4.5.3.

4.5.1 Architecture

We build Dimmer over LWB's 2019 reimplementation [143], and depict its architecture in Fig. 4.6. Dimmer is composed of four main novel components: a *statistics collector*, used to close the feedback loop, the *central adaptivity control* incorporating our deep-Q network (DQN), the *forwarder selection* implementing

multi-armed bandits, and a *controller* managing and coordinating the different components. The statistics collector has privileged access to the LWB runtime and sent and received packets. The controller updates internal scheduling parameters as well as the bus interface (i.e., retransmission) parameters. At the end of a communication round, the controller queries the central adaptivity control for new parameters, and let nodes autonomously select non-critical forwarders with their forwarder selection scheme if no losses have been detected.

Feedback loop. Like LWB, Dimmer works in communication rounds (c.f. Fig. 4.3). A central coordinator acts as the LWB host. Each communication round starts with a schedule dissemination from the coordinator. In addition to the LWB schedule, Dimmer incorporates an adaptivity control command, e.g., a new value N for the retransmission parameter. As soon as the schedule slot has finished, all nodes apply the new parameter. If no losses have been detected, the coordinator let nodes run their distributed forwarder selection scheme.

A series of attributed data slots follows the first schedule dissemination. For each data slot, the source node appends to its payload a four-byte metadata representing two performance metrics: its radio-on time averaged over the last floods, and its reliability. Whenever such a data packet is received, its metrics are locally recorded by all receivers. This aggregation of performance metrics allows Dimmer to create a global snapshot of the system's behavior, and is used by both the central adaptivity control at the coordinator and the forwarder selection scheme at the nodes. We estimate the reliability via the schedule: if no message is received during an assigned slot, it is considered lost. If no information is received from a given node, the network assumes the worst and sets its statistics accordingly (100% radio-on time, 0% reliability). Thus, missing metrics are always pessimistically filled.

Latency of feedback. Collecting feedback is not an instantaneous procedure. While the retransmission parameter is updated at the beginning of the communication round T, the effectiveness of the update is only known at the end of round $T+1$: node first locally measure performance, and later share their statistics during the next round. Thus, collecting feedback always take two communication rounds, assuming enough data slots are available during a round. We note that although Dimmer's latency has no strong dependency on the number of nodes, the round periodicity might increase for large deployments.

4.5.2 Central Adaptivity Control

Under challenging wireless conditions, it is imperious to maintain the maximum achievable reliability. We endow Dimmer with a central policy control unit, operating an embedded deep Q-network, see Fig. 4.3. Whenever interference is detected, Dimmer updates the retransmission parameter to tackle the performance degradation, and reduces it once the interfering episode has passed. Since LWB communication starts with a schedule dissemination from the coordinator, Dimmer adapts the retransmission parameter of the entire network at no additional communication cost, by simply piggybacking commands with the schedule.

Embedded DQN. We target TelosB motes as platform, meaning we restrict ourselves to a 4 Mhz 16-bit CPU with 10KB of RAM. In §4.4.1, we

specify our DQN with two fully connected layers: a hidden layer of 30 neurons with ReLU, and an output layer of three neurons. Due to the simplicity of our neural architecture and in order to save space, we implement our own neuronal compute-system rather than use an existing framework. Like other embedded solutions [144], we use fixed-point integers for computation. We choose a fixed point of 100, effectively truncating our initial weights to two floating digits and representing them as integers. By using two bytes per weight and four bytes for intermediary computation, our DQN uses around 2.5 KB of memory: 2.1 KB are used to store weights in flash, while 400 B of RAM are used for intermediary results at runtime. Obtaining an output takes roughly 90 ms. This high latency is due to the use of 32 bits integer for computation on a 16-bit architecture.

4.5.3 Distributed Forwarder Selection

In the absence of interference, not all transmissions in a Glossy flood are beneficial [132, 133]. Nodes can be classified as active forwarders, acting as packet relays during the flood, or as leaf nodes, for which transmissions are non-critical for the correct dissemination of the flood. By deactivating leaf nodes, it is possible to save energy without degrading reliability.

Serialized Exp3. We model the forwarder selection problem as a multi-armed bandit (MAB) problem and solve it using Exp3 [141]. Each node executes a local instance of Exp3. If the central coordinator does not detect interference, it lets the nodes execute their forwarder selection scheme. To avoid concurrent updates, at most one node learns at a time. We select the learning node following a globally known, pseudo-random sequence. During the learning stage, the node randomly decides whether to act as forwarder or leaf, drawn from a probability computed by Exp3. Once feedback is collected, the learning node is rewarded or punished if performance is degraded. In addition, if the node causes loss by acting as a leaf, we further reinforce the 'forwarder' arm to avoid further losses. After 20 learning iterations, the learning node freezes its current behavior, and the next node in the sequence starts learning.

Slow adaptivity. Convergence to the optimal state is not guaranteed. Systems using Exp3 can suffer non-convergent states, where action weights oscillate [93]. Rewarding the 'forwarder' arm after losses, using a globally-known pseudo-random sequence and giving 20 consecutive learning iterations are key techniques we use to reduce the risk of non-convergence. In addition, such distributed instances can be slow to react. For a deployment with n, we can require up to $20 \times n$ rounds to reach a new, stable forwarder set.

4.6 Evaluation

We evaluate Dimmer on two testbeds composed of 18 and 48 TelosB nodes, respectively. We investigate the following questions: Which input features affect the overall performance of Dimmer (§4.6.2), How does adaptivity manifest (§4.6.3), How does the distributed forwarder selection perform (§4.6.4), and can Dimmer be used on a different topology with no additional training (§4.6.5).

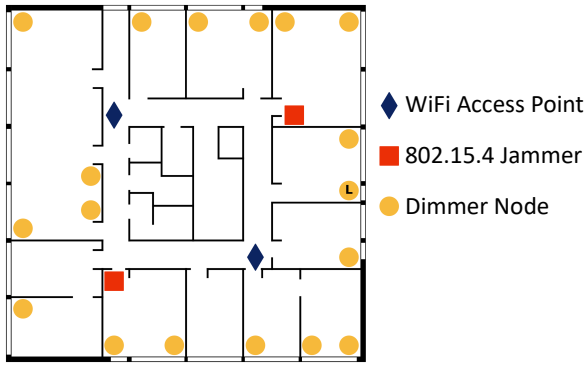


Figure 4.7. Testbed topology: 18 resource-constrained TelosB nodes are deployed throughout offices and student lab rooms. The deployment is colocated with two WiFi access points, and has a 3-hop network diameter. Two IEEE 802.15.4 jammers are placed to interfere as many nodes as possible.

4.6.1 Setup and Methodology

We lay out the setup we use in this evaluation, as well as our experimental methodology we follow.

Implementation. Based on LWB's reimplementation for Contiki-NG [143], we build Dimmer for the TelosB platform featuring a 4 MHz 16-bit CPU, 10 KB of RAM, 48 KB of firmware storage, and a CC2420 radio compatible with IEEE 802.15.4. We implement Dimmer in C, and we make the primitive and its training environment open-source (see footnote in §4.1 for a link).

Testbeds. We evaluate Dimmer on a local testbed comprising 18 TelosB motes, see Fig. 4.7. The testbed is deployed in offices and students lab rooms, and shares the spectrum with WiFi deployments and many Bluetooth PANs (from cellphones, headphones, keyboards etc.). Both are outside of our control and commonly are the source of heavy traffic, i.e., interference, during work hours. Additionally, two TelosB motes are used as jammers to inject 802.15.4 interference using Jamlab [145], in a controlled manner.

In the second part of the evaluation, we evaluate Dimmer on the public testbed D-Cube, featuring 48 TelosB motes and the ability to inject WiFi interference [146].

PI controller baseline. To investigate the advantages of RL, we additionally evaluate a traditional solution for closed-loop systems, a PI controller [136], enhanced with handcrafted rules. We set K_P to 1 and K_I to 0.25, and add additional mechanisms to the controller to avoid rapid parameter decrease, which inevitably leads to decreased reliability if done too often. We test and calibrate our PI controller both with the traces we use to train Dimmer, as well as on the deployment itself. We calibrate the controller to provide ∼99.9% reliability under no interference, and to maintain high reliability even under jamming.

Interference scenarios. The testbed used is colocated with WiFi and Bluetooth PANs. We evaluate the following scenarios:

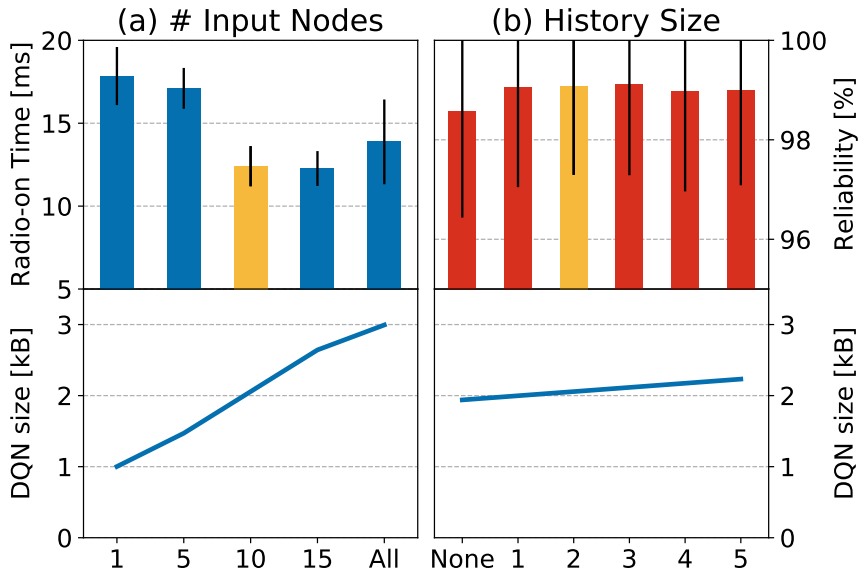


Figure 4.8. Impact of the number of nodes and historical data as input to the DQN. (a) Using only the worst node performance as input to the DQN leads to an overly conservative strategy that wastes energy but does not improve reliability. (b) Using historical data helps improve reliability. We select 10 input nodes and 2 historical features.

- No interference: experiments ran at night on channel 26 (i.e., without WiFi), no injected interference.
- Daytime level (uncontrolled interference): experiments ran during the day on channel 13, with colocated WiFi and Bluetooth PANs.
- Controlled 802.15.4 interference: We use the CC2420 radio chip's ability of generating a pseudo-random data sequence that we transmit at 0 dBm on additional TelosB motes, on channel 26. We interfere with the medium with a 13 ms burst, which corresponds to a typical WiFi burst of packets [145], followed by a calm period. We quantify the interference strength by the ratio of time the channel is jammed: an interference of 10% corresponds to a 13 ms burst followed by 117 ms with no interference, while a 35% interference ratio represents a 13 ms burst followed by 24 ms calm period.
- D-Cube [146]: We use the public testbed D-Cube, featuring controlled WiFi interference.

Metrics. Throughout this section, we use the two metrics, defined as follows:

- Radio-on time: the amount of time the radio has been active (either listening or transmitting) for one slot. The radio-on time is averaged over

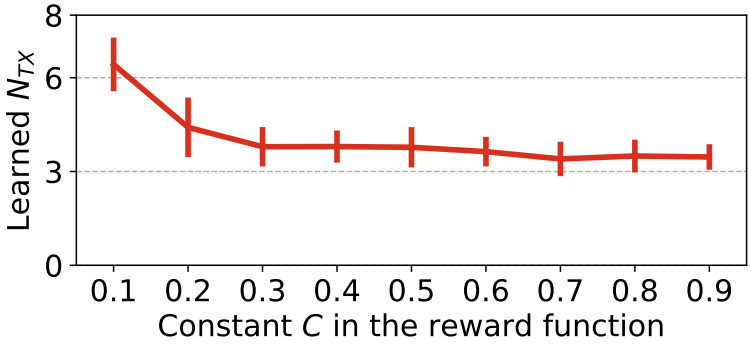


Figure 4.9. Tuning the reward function: increasing C in Eq. 4.3 has a direct effect on the learned strategy. Low C values favor reliability over energy, while high values improve energy efficiency.

all slots since the last parameter update. Slots in which no packet was received are accounted for.

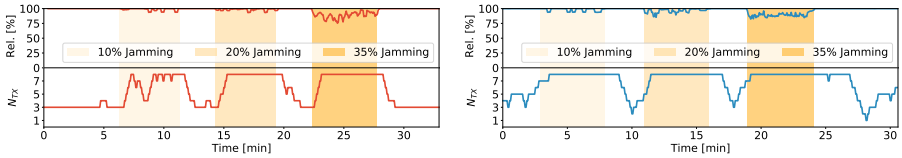
- Network reliability: application-level reliability, a packet is considered lost if at least one destination did not receive it.

4.6.2 Deep-Q Network Features Selection

In this section, we study the effect of the number of historical and performance features used as input to Dimmer, as well as the impact of the reward function on the overall performance. We collect a evaluation dataset of 25 000 samples over channel 26, featuring some periods of mild and heavy interference, and some interference-free periods. For each parameter we evaluate, we train three models, and average the overall performance over those models. For each model, we run 100 episodes comprising 100 consecutive decisions each for the number of nodes and reward, and 1000 episodes of 2 consecutive decisions to show the direct effect of historical features.

Number of Nodes as Input: Scenario. We evaluate how many node inputs are necessary for Dimmer. As described in detail in §4.4.1, Dimmer selects K nodes with the worst performance as input to its DQN. We fix the number of historical features to 2, and vary K from one, i.e., only the worst performance is accounted, up to all nodes are used as input.

Results. Fig. 4.8a depicts the effect of K on Dimmer's radio-on time. Taking a very limited subset of nodes leads to an overly conservative policy, wasting energy by retransmitting too often, even in non-interfered scenarios. With all nodes used as input, the DQN tends to overfit, and reacts even to transient disturbances, as depecited by the high radio-on time in Fig. 4.8a. Note that the reliability is constant for all experiments, meaning that these overly conservative strategies do not improve further reliability. For the remainder of this evaluation, we choose $K = 10$ as our input, which both minimizes the radio-on time, as well as provides a good trade-off w.r.t. the neural network



(a) Dimmer against injected interference. Dimmer effectively detects interference and quantifies its severity, increasing its retransmissions to maintain high reliability. Dimmer then converges back to the optimal parameter for the non-interfered medium.

(b) PI-based controller against injected interference. The PI controller oscillates around the optimal parameter for the non-interfered medium and over-provisions as soon as interference is detected, irrespective of its severity, leading to wasted energy.

Figure 4.10. Adaptiveness to interference.

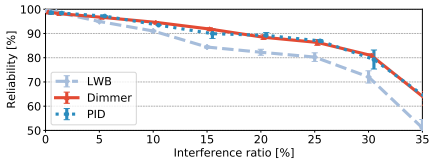
size.

History Size: Scenario. We evaluate if historical features are beneficial to adaptivity. In §4.4.1, we define an historical feature as 1 if no losses were detected the previous round, and -1 if at least one packet was lost. For this scenario, we focus on short decision updates in low and mild interfered environment, to test Dimmer's ability to distinguish transient disturbances from longer-term interference.

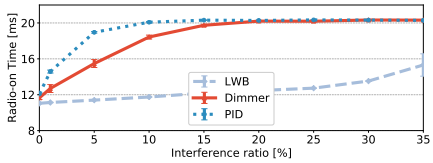
Results. Fig. 4.8b shows the impact of using historical features. Adding historical features helps Dimmer differentiate transient interference that affects a single round, from long-term interference that must be dealt with. In the absence of historical features, the DQN obtains 98.5% reliability, while it achieves 99% with historical features. The number of historical features does not seem to have a measurable impact on the overall performance in our evaluation, but the interference patterns present in the training dataset can affect the overall impact of historical features. As our training set contains traces where we inject perturbations to the wireless medium followed by long periods of calm, a single historical feature is sufficient to detect interfered-periods from transient disturbances. Other interference patterns might require more historical features to be correctly detected. For the remainder of this evaluation, Dimmer uses two historical features.

Reward Function: Scenario. We evaluate the effect of the constant C in our reward function, see Eq; 4.3, over both interfered and non-interfered environments.

Results. Fig. 4.9 depicts the impact of C in the reward function. For a low constant C, the system learns to use a high retransmission parameter, even in non-interfered environments. From $C = 0.3$ and higher, Dimmer converges to $N_{TX} = 3$, the value recommended by Glossy's designers [23]. Note that for high C values, the Q-function learned by the DQN outputs small values, which can negatively impacts performance once the DQN is quantized. We select $C = 0.3$ for the remainder of this evaluation.



(a) Reliability against interference. Both Dimmer and the PI controller are able to tackle increasing interference by adapting their retransmission factor. Because some slots manage to fit between two interference bursts, LWB is able to maintain some communication.



(b) Radio-on time against interference. The PI controller quickly reacts to interference and overprovision retransmissions, leading to a maximized energy consumption. In contrast, Dimmer adapts to the interference level, leading to a lower radio-on time compared to the PID, for similar reliability.

Figure 4.11. Resilience to interference.

4.6.3 Adaptivity Against Interference

Next, we evaluate how Dimmer adapts to interference. First, we depict how Dimmer handles coming-and-going interference patterns. Then, we evaluate Dimmer's performance against different interference levels.

Scenario: dynamic interference. We operate Dimmer on our office-deployed testbed, see §4.6.1, on channel 26. After five minutes, we inject IEEE 802.15.4 interference using additional TelosB motes. We repeat this injected interference followed by a calm period three times, with varying interference strength, using 10, 20 and 35% jamming, respectively representing low, mild and heavy interference. We then repeat the same scenario, replacing our DQN with a PI controller enhanced with handcrafted rules (c.f. §4.6.1)

Results. Fig. 4.10a depicts Dimmer's operation against dynamic interference. We depict reliability using a smoothing window of size $s = 5$. As soon as losses are detected, Dimmer increases its retransmission factor N_{TX} to tackle the disturbance. Once the interfering episode has passed, Dimmer quickly converges back to its default, learned value. Note that Dimmer is able to differentiate low interference (10%), where it tries to reduce N_{TX} more often, from heavier interference, where the maximum retransmission is maintained longer. While the PI controller is also able to detect interference and quickly counter its effects (Fig. 4.10b), it is unable to distinguish different interference patterns, and we have to supplement it with handcrafted rules to avoid decreasing too early. Furthermore, our PI controller is unable to find the optimal N_{TX} in during interference-free periods, and oscillates constantly. Note that although the PI controller oscillates, we tuned its parameters to provide almost 100% reliability in the interference-free case. It is possible to improve the controller's behavior, but such tuning would be time-consuming. In the depicted run, Dimmer achieves an average reliability of 98.1%, and the PI controller obtains 97.9%, for a much higher radio-on time, meaning Dimmer is well suited to detect and adapt to different interference patterns.

Scenario: interference levels. We run Dimmer for 30 minutes against a continuous, static interference-pattern, on channel 26. We evaluate Dimmer's

reliability and the amount of time the radio is active for, for different interference levels. We repeat this scenario for the PI controller described above, as well as for LWB, with its default retransmission parameter $N_{TX} = 3$. Results are averaged over three 30-minute runs for each interference level.

Results. Fig. 4.11a depicts Dimmer, LWB and the PI controller's reliabilities against static interference, while Fig. 4.11b shows their radio-on time as a proxy for energy consumption. While all protocols performance decreases as interference arises, both Dimmer and the PI controller are able to provide similar, higher reliability. As the PI controller is unable to distinguish interference levels, its radio-on time reaches the maximum slot length of 20 ms even in low-interfered environments. In contrast, Dimmer is able to quantify interference strength, and reaches the maximum slot length at 20% interference only, and is thus able to save energy compared to the handcrafted controller, while offering the same performance.

In conclusion, it is possible to design a solution, either using a PI controller or handcrafted rules, to adapt to interference. However, tuning it to differentiate interference levels is arduous and a research task in itself, requiring expertise in the field. In contrast, Dimmer is able to learn how to differentiate interference levels and how to adapt from unlabeled traces, and does not require expert knowledge during deployment. From this point, we do not evaluate further the performance of our PI controller.

4.6.4 Forwarder Selection with MAB

Next, we evaluate the second component of Dimmer, its distributed forwarder selection scheme using multi-armed bandits.

Scenario. We execute Dimmer's forwarder selection scheme on channel 26 for 10 hours. During that time, the DQN is deactivated to avoid interference in the learning process. Each node is given 20 consecutive rounds to try whether it should act as forwarder, see §4.4.3 for details. We evaluate the number of nodes that act as forwarders throughout time, as well as the impact of learning on performance.

Results. Fig. 4.12 depicts the network reliability and the number of active forwarders through time. Within the first hour, many nodes stop acting as forwarders. As reliability drops in bad configurations, nodes adapt and act again as forwarders, for example here at the 1 and 7-hour marks. After 8 hours, the network stabilizes with 10 active forwarders out of 18. On average, nodes are active for 8.7 ms, while they are on average active for 11 ms without forwarder selection. Because leaf nodes still listen at the beginning of a slot, synchronisation does not break for leaf nodes. Fig. 4.13 depicts the learning process of a single forwarder. It must be noted that since we cannot guarantee any convergence, some nodes will change their role, possibly multiple times, during the protocol execution. Neighboring nodes will adapt accordingly to maintain communication.

4.6.5 Performance on Unknown Deployments

In this section, we evaluate whether Dimmer is able to perform on different deployments, against unseen interference, without retraining its DQN.

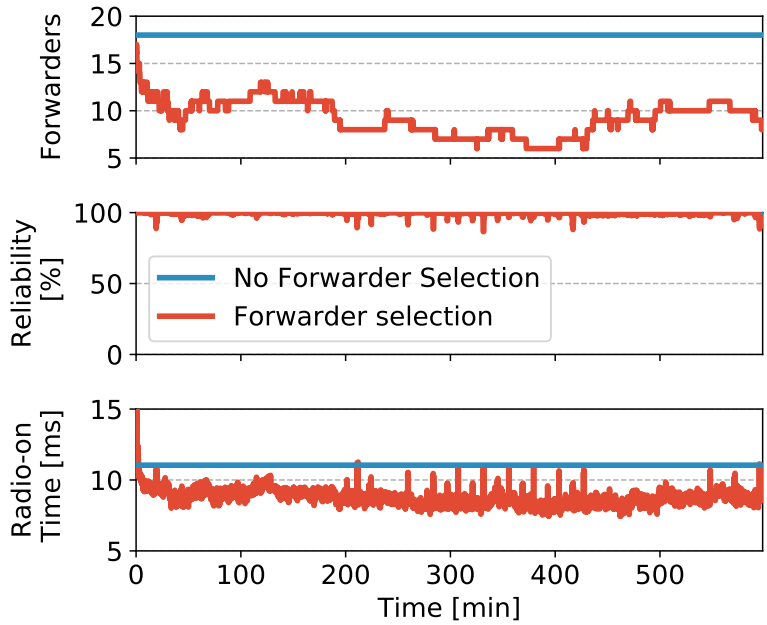


Figure 4.12. Forwarder Selection with Multi-Armed Bandits. Nodes takes turn learning whether to act as forwarder. As learning occurs, the number of forwarders decreases, but learning might cause packet losses.

Scenario. We execute Dimmer on D-Cube, a public testbed featuring 48 TelosB nodes [146]. D-Cube is able to inject either 802.15.4 or WiFi interference. We limit Dimmer to channel 26 to evaluate the generality of adaptivity. We run Dimmer in three scenarios: an interference-free case, a 802.15.4 interference case, and a WiFi interference case. As the injected WiFi interference surpasses any 15.4 transmission, no single-channel protocol is able to ensure reliable delivery in this scenario. The WiFi and 802.15.4 interference in D-Cube are reproducible, and we consider the first level of interference available in this evaluation. We use the Data Collection V1 scenario defined in D-Cube. In this scenario, a known set of five nodes transmits packets at random intervals to a known sink. Here, reliability is defined as the number of packets received at the sink. For each scenario, we run five 10-minute experiments and average the results.

Results. Fig. 4.14 depicts the results of Dimmer and LWB on D-Cube for all three scenarios. Both LWB and Dimmer obtain 100% reliability in the interference-free case, for similar energy consumption. Under 802.15.4 interference, LWB and Dimmer obtain 99.4% and 99.6% reliability respectively. In this scenario, interference seems to be sparse, and Dimmer hardly differentiates from normal, transient disturbances. Finally, we inject WiFi interference against Dimmer. LWB higly suffers from the disturbance, and obtains on average 30% reliability. In contrast, Dimmer is able to achieve 58% reliability on average,

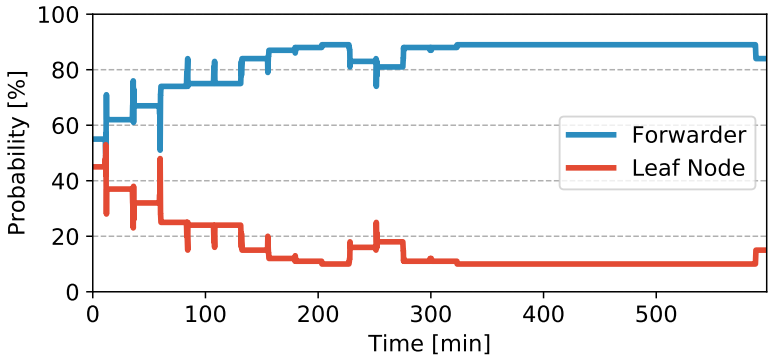


Figure 4.13. Probability to act as forwarder. As time passes, a node learns it is vital for a correct flood propagation. A small probability is maintained to allow adaptation to new network dynamics.

reaching 80% on multiple instances, but also dropping to 30% in one run, even if Dimmer does recognize the interference and adapts its retransmission to $N_{TX} = 8$ in all runs. This means that Dimmer is able to perform on a new topology with more than two times the number of devices, as well as react to interference never seen before, without the need to retrain its DQN.

4.7 Related Work

Surviving interference. Traditional approaches to improving reliability focus on two orthogonal techniques: transmission scheduling and channel hopping. Coupled with a higher, fixed retransmission count, Robust Flooding successively transmits the packet several times in a row in order to improve resilience to interference [32]. In contrast, we update the retransmission count to adapt to current interference levels in Dimmer, but keep Glossy's transmission schedule. Going multi-channel is also an effective and well documented approach to survive interference and jamming: Istomin et al. improve Crystal, an aperiodic data collection primitive, against heavy interference scenarios using channel-hopping [28]. Similarly, the winning solution for the 2019 EWSN dependability competition, DeCoT+ [134], uses multi-channel, as well as most solutions in the previous years. Dimmer supports a simple slot-based channel-hopping scheme, but we plan to improve it with a learned blacklisting method in the future.

Traffic-based adaptivity. Earlier works refer to adaptivity as the ability to adapt traffic changes. LWB is able to adapt its round periodicity as well as the number of data slots to the current traffic needs [27]. Blink builds on top of LWB to provide communication with end-to-end guarantees, by building a schedule at runtime [80]. In contrast, we use adaptivity in this paper to refer to the ability to adapt to environment changes.

AI-enabled wireless systems. Lately, RL and deep-RL (i.e., RL using deep neural networks) have been used to solve communication problems: whether it is for routing [147], estimating link quality for RPL [148], MAC

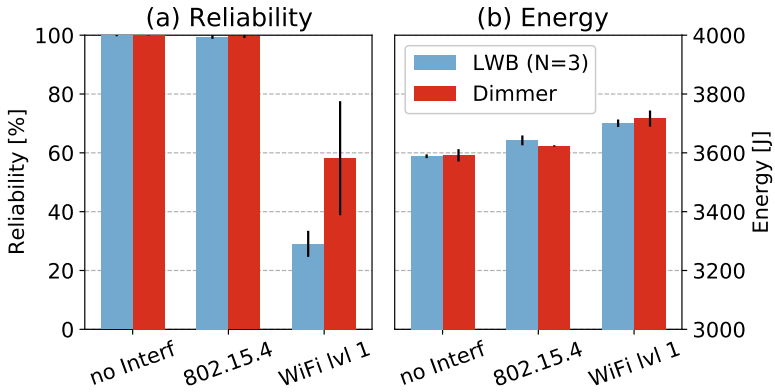


Figure 4.14. Porting Dimmer to a new deployment featuring unknown interference. Without retraining, Dimmer is able to tackle previously unseen WiFi patterns, on a new 48-node deployment.

contention [91], schedule transmission [149], channel selection [92], or even learning new modulation schemes [59].

Closer to our work, Zhang et al. use multi-armed bandits (MAB) to select the number of retransmission within a Glossy flood [93]. Zhang et al. set out to find the optimal N parameter for each node in the network, but assume that (1) the environment does not evolve quickly, and (2) possibly break the constructive interference requirements when providing feedback. In contrast, our work uses a DQN to find the global N parameter under varying conditions, and uses MAB to find a subset of necessary forwarders and leaf nodes to save energy during interference-free periods.

4.8 Conclusion

Although Glossy-based communication primitives provide low-latency and high-reliability message delivery, they suffer from two limitations: (1) they are invariant to their environment dynamics, and rely on over-provisioning to maintain performance under disturbances, and (2) their one-fits-all design lead to wasted energy in dense deployments. We argue that communication primitives for low-power wireless networks should be more adaptive with respect to their environment, by meeting a target performance while ensuring energy efficiency. We introduce Dimmer, an adaptive all-to-all communication protocol built on top of LWB, bringing the recent advances of deep reinforcement learning to low-power networking. We combine a deep Q-network that centrally controls the retransmission parameter to tackle quick interference surges, with a distributed multi-armed bandits approach to save energy during interference-free periods. Trained over unlabeled traces, Dimmer adapts to perturbations to the medium, and can be ported to different deployments without the need to retrain.

In the future, we plan to extend Dimmer with blacklisting-enabled multi-

channel capabilities, where Dimmer will learn to avoid and reintroduce channels based on past performance.

Bibliography

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[2] Z. Fu, O. Landsiedel, M. Almgren, and M. Papatriantafilou, "Managing your trees: Insights from a metropolitan-scale low-power wireless network," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM) Workshops*. IEEE, 2014, pp. 664–669.

[3] M. Petracca, S. Bocchino, A. Azzara, R. Pelliccia, M. Ghibaudi, and P. Pagano, "WSN and RFID integration in the IoT scenario: an advanced safety system for industrial plants," *Journal of communications software and systems*, vol. 9, pp. 104–113, 2013.

[4] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in ZebraNet," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2004, pp. 227–238.

[5] X. Mao, X. Miao, Y. He, X. Li, and Y. Liu, "CitySee: Urban CO2 monitoring with sensors," *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1611–1619, 2012.

[6] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.

[7] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier *et al.*, "PermaDAQ: A scientific instrument for precision sensing and data recovery in environmental extremes," in *Proceedings of the IEEE/ACM International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, April 2009, pp. 265–276.

[8] M. Erdelj, E. Natalizio, K. R. Chowdhury, and I. F. Akyildiz, "Help from the sky: Leveraging UAVs for disaster management," *IEEE Pervasive Computing*, vol. 16, no. 1, pp. 24–32, Jan 2017.

[9] E. A. Lee, "Cyber physical systems: Design challenges," in *Proceedings of the IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, May 2008, pp. 363–369.

[10] I. Stoianov, L. Nachman, S. Madden *et al.*, "PIPENET: A wireless sensor network for pipeline monitoring," in *Proceedings of the IEEE/ACM International Symposium on Information Processing in Sensor Networks (IPSN)*. IEEE, April 2007, pp. 264–273.

[11] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2004, pp. 95–107.

[12] A. Dunkels, "The contikimac radio duty cycling protocol," SICS, Tech. Rep., 2012.

[13] R. Min, M. Bhardwaj, Seong-Hwan Cho, E. Shih, A. Sinha, A. Wang, and A. Chandrakasan, "Low-power wireless sensor networks," in *Proceedings of the International Conference on VLSI Design*, 2001, pp. 205–210.

[14] M. Ceriotti, M. Corr, L. D'Orazio, R. Doriguzzi, D. Facchin, S. T. Gun, G. P. Jesi, R. L. Cigno, L. Mottola, A. L. Murphy, M. Pescalli, G. P. Picco, D. Pregnolato, and C. Torghele, "Is there light at the ends of the tunnel? wireless sensor networks for

adaptive lighting in road tunnels," in *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2011, pp. 187–198.

[15] M. Ferreira, R. Fernandes, H. Conceição, W. Viriyasitavat, and O. K. Tonguz, "Self-organized traffic control," in *Proceedings of the ACM International Workshop on VehiculAr InterNETworking (VANET)*. ACM, 2010, pp. 85–90.

[16] "802.15.4-2015: IEEE standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (LR-WPANs) 1: MAC sublayer."

[17] Bluetooth SIG, "Bluetooth core specification (revision 5.2)," 2019.

[18] "LoRa alliance," https://lora-alliance.org/, accessed: 2020-07-15.

[19] "Nordic semi nrf52840 platform," https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840, accessed: 2020-05-04.

[20] "STM32 wireless MCUs," https://www.st.com/en/microcontrollers-microprocessors/stm32-wireless-mcus.html, accessed: 2020-05-04.

[21] C. A. Boano, "Session details: Dependability competition," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2019.

[22] S. Duquennoy, B. Al Nahas, O. Landsiedel *et al.*, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, 2015, pp. 337–350.

[23] F. Ferrari, M. Zimmerling, L. Thiele *et al.*, "Efficient network flooding and time synchronization with Glossy," in *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2011.

[24] M. Bashiri and C. H. Fleming, "A platoon-based intersection management system for autonomous vehicles," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 667–672.

[25] B. Li, L. Nie, C. Wu, H. Gonzalez, and C. Lu, "Incorporating emergency alarms in reliable wireless process control," in *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. ACM, 2015, pp. 218–227.

[26] A. Köpke, "Engineering a communication protocol stack to support consensus in sensor networks," Ph.D. dissertation, TU Berlin, 2012.

[27] F. Ferrari, M. Zimmerling, L. Mottola *et al.*, "Low-power wireless bus," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2012, pp. 1–14.

[28] T. Istomin, M. Trobinger, A. L. Murphy *et al.*, "Interference-resilient ultra-low power aperiodic data collection," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2018, pp. 84–95.

[29] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Virtual synchrony guarantees for cyber-physical systems," in *Proceedings of the IEEE International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, Sep. 2013, pp. 20–30.

[30] A. Jhumka and L. Mottola, "Neighborhood view consistency in wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 12, no. 3, Jul. 2016.

[31] K. Srinivasan, M. A. Kazandjieva, S. Agarwal *et al.*, "The β-factor: Measuring wireless link burstiness," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2008, pp. 29–42.

[32] R. Lim, R. Da Forno, F. Sutton *et al.*, "Competition: Robust flooding using back-to-back synchronous transmissions with channel-hopping," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2017, pp. 270–271.

[33] "Sigfox," https://www.sigfox.com/en, accessed: 2020-07-15.

[34] M. Ringwald and K. Römer, "BitMAC: a deterministic, collision-free, and robust mac protocol for sensor networks," in *Proceedings of the European Workshop on Wireless Sensor Networks (EWSN)*, Feb 2005, pp. 57–69.

[35] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis, "Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2010, pp. 1–14.

[36] M. Wilhelm, V. Lenders, and J. B. Schmitt, "On the reception of concurrent transmissions in wireless sensor networks," *IEEE Transactions on Wireless Communications (TWC)*, vol. 13, no. 12, pp. 6756–6767, 2014.

[37] C. Liao, Y. Katsumata, M. Suzuki, and H. Morikawa, "Revisiting the so-called constructive interference in concurrent transmission," in *Proceedings of the IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2016, pp. 280–288.

[38] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Concurrent transmissions for multi-hop Bluetooth 5," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2019, pp. 130–141.

[39] M. Baddeley, C. A. Boano, A. Escobar-Molero, Y. Liu, X. Ma, U. Raza, K. Römer, M. Schuß, and A. Stanoev, "The impact of the physical layer on the performance of concurrent transmissions," *arXiv preprint arXiv:2005.13816*, 2020.

[40] D. Lobba, M. Trobinger, D. Vecchia, T. Istomin, and G. P. Picco, "Concurrent transmissions for multi-hop communication on ultra-wideband radios," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2020, pp. 132–143.

[41] K. Leentvaar and J. Flint, "The capture effect in FM receivers," *IEEE Transactions on Communications*, 1976.

[42] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*, 2013.

[43] C. Herrmann, F. Mager, and M. Zimmerling, "Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, 2018.

[44] M. Mohammad and M. C. Chan, "Codecast: Supporting data driven in-network processing for low-power wireless sensor networks," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. ACM, 2018.

[45] M. Zimmerling, L. Mottola, and S. Santini, "Synchronous transmissions in low-power wireless: A survey of communication protocols and network services," *ArXiv*, vol. abs/2001.08557, 2020.

[46] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, 1985.

[47] A. D. Kshemkalyani and M. Singhal, *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.

[48] E. A. Brewer, "Towards robust distributed systems," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2000, p. 7.

[49] J. Gray, "Notes on data base operating systems," in *Operating Systems, An Advanced Course*, 1978.

[50] D. Skeen and M. Stonebraker, "A formal model of crash recovery in a distributed system," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 3, 1983.

[51] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, 1998.

[52] ——, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column)*, vol. 32, pp. 51–58, December 2001.

[53] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[54] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. MIT Press, 1998.

[55] C. C. White, *Markov decision processes*. Springer US, 2001, pp. 484–486.

[56] R. E. Bellman, *Dynamic Programming*. Dover Publications, 2003.

[57] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015.

[58] N. Lazic, C. Boutilier, T. Lu *et al.*, "Data center cooling using model-predictive control," in *Advances in Neural Information Processing Systems 31 (NeurIPS)*. Curran Associates, 2018, pp. 3814–3823.

[59] C. de Vrieze, S. Barratt, D. Tsai *et al.*, "Cooperative multi-agent reinforcement learning for low-level wireless communication," *arXiv e-prints*, Jan 2018.

[60] N. Mastronarde, J. Modares, C. Wu *et al.*, "Reinforcement learning for energy-efficient delay-sensitive CSMA/CA scheduling," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, Dec 2016, pp. 1–7.

[61] L. Paradis and Q. Han, "A survey of fault management in wireless sensor networks," *Journal of Network and systems management*, vol. 15, no. 2, pp. 171–190, 2007.

[62] L. B. Ruiz, I. G. Siqueira, L. B. e. Oliveira, H. C. Wong, J. M. S. Nogueira, and A. A. F. Loureiro, "Fault management in event-driven wireless sensor networks," in *Proceedings of the ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. ACM, 2004, pp. 149–156.

[63] X. Miao, K. Liu, Y. He, D. Papadias, Q. Ma, and Y. Liu, "Agnostic diagnosis: Discovering silent failures in wireless sensor networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 12, pp. 6067–6075, 2013.

[64] A. Jhumka and L. Mottola, "On consistent neighborhood views in wireless sensor networks," in *Proceedings of the IEEE International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2009, pp. 199–208.

[65] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2004, pp. 13–24.

[66] J. Paek and R. Govindan, "Rcrt: Rate-controlled reliable transport protocol for wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 7, no. 3, Oct. 2010.

[67] S. Kumar, M. P. Andersen, H.-S. Kim, and D. E. Culler, "Performant TCP for low-power wireless networks," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Feb. 2020, pp. 911–932.

[68] R. Jacob, L. Zhang, M. Zimmerling, J. Beutel, S. Chakraborty, and L. Thiele, "TTW: A time-triggered wireless design for CPS," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 865–868.

[69] O. Chipara, C. Wu, C. Lu, and W. Griswold, "Interference-aware real-time flow scheduling for wireless sensor networks," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, 2011, pp. 67–77.

[70] A. Benchi and P. Launay, "Solving Consensus in Opportunistic Networks," in *International Conference on Distributed Computing and Networking (ICDCN)*. ACM, 2015.

[71] G. Chockler, M. Demirbas, S. Gilbert, C. Newport, and T. Nolte, "Consensus and collision detectors in wireless ad hoc networks," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2005.

[72] H. Moniz, N. F. Neves, and M. Correia, "Byzantine fault-tolerant consensus in wireless ad hoc networks," *IEEE Trans. on Mobile Computing*, vol. 12, no. 12, 2013.

[73] F. Borran, R. Prakash, and A. Schiper, "Extending paxos/lastvoting with an adequate communication layer for wireless ad hoc networks," in *IEEE SRDS*, 2008.

[74] C. A. Boano, M. A. Zúñiga, K. Römer, and T. Voigt, "JAG: Reliable and predictable wireless agreement under external radio interference," in *IEEE RTSS*, 2012.

[75] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Network-wide consensus utilizing the capture effect in low-power wireless networks," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2017.

[76] A. Spina, M. Breza, N. Dulay, and J. McCann, "XPC: Fast and reliable synchronous transmission protocols for 2-phase commit and 3-phase commit," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2020.

[77] A. Spina, "Reliable distributed consensus for low-power multi-hop networks," Master's thesis, Imperial College London, 2019.

[78] L. Hobert, A. Festag, I. Llatser, L. Altomare, F. Visintainer, and A. Kovacs, "Enhancements of V2X communication in support of cooperative autonomous driving," *Infocommunications Journal*, vol. 8, no. 3, pp. 27–33, 2016.

[79] K. Dresner and P. Stone, "A Multiagent Approach to Autonomous Intersection Management," *Journal of Artificial Intelligence Research*, vol. 31, pp. 591–656, 3 2008.

[80] M. Zimmerling, L. Mottola, P. Kumar *et al.*, "Adaptive real-time communication for wireless cyber-physical systems," *ACM Transactions on Cyber-Physical Systems*, vol. 1, no. 2, Feb. 2017.

[81] S. Munir, S. Lin, E. Hoque, S. M. S. Nirjon, J. A. Stankovic, and K. Whitehouse, "Addressing burstiness for reliable communication and latency bound generation in wireless sensor networks," in *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. ACM, 2010, pp. 303–314.

[82] C. Noda, S. Prabh, M. Alves, C. A. Boano, and T. Voigt, "Quantifying the channel quality for interference-aware wireless sensor networks," *SIGBED Rev.*, vol. 8, no. 4, pp. 43–48, Dec. 2011.

[83] L. Mottola and G. P. Picco, "MUSTER: Adaptive energy-aware multisink routing in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 10, no. 12, pp. 1694–1709, Dec. 2011.

[84] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the International Workshop on Internet of Things towards Applications (IoT-App)*. ACM, 2015, pp. 7–12.

[85] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2016, pp. 176–189.

[86] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2016.

[87] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[88] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2017.

[89] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "FastDeepIoT: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2018, pp. 278–291.

[90] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2 KB RAM for the internet of things," in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 70. PMLR, Aug 2017, pp. 1935–1944.

[91] S. Amuru, Y. Xiao, M. van der Schaar *et al.*, "To send or not to send - learning mac contention," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, Dec 2015, pp. 1–6.

[92] H. Dakdouk, E. Tarazona, R. Alami *et al.*, "Reinforcement learning techniques for optimized channel hopping in IEEE 802.15.4-TSCH networks," in *Proceedings of the ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*. ACM, 2018, pp. 99–107.

[93] P. Zhang, A. Y. Gao, and O. Theel, "Less is more: Learning more with concurrent transmissions for energy-efficient flooding," in *Proceedings of the EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*. ACM, 2017, pp. 323–332.

[94] F. Restuccia and T. Melodia, "DeepWiERL: Bringing Deep Reinforcement Learning to the Internet of Self-Adaptive Things," *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2020.

[95] S. Joseph, R. Misra, and S. Katti, "Towards self-driving radios: Physical-layer control using deep reinforcement learning," in *Proceedings of the International Workshop on Mobile Computing Systems and Applications (HotMobile)*. ACM, 2019, pp. 69–74.

[96] V. Poirot, B. A. Nahas, and O. Landsiedel, "Paxos made wireless: Consensus in the air," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2019, pp. 1–12.

[97] P. Rathje, V. Poirot, and O. Landsiedel, "STARC: Low-power decentralized coordination primitive for vehicular ad-hoc networks," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2020, pp. 1–6.

[98] I. Maza, F. Caballero, J. Capitán, J. R. Martínez-de-Dios, and A. Ollero, "Experimental results in multi-UAV coordination for disaster management and civil security applications," *J. of Intelligent & Robotic Systems*, vol. 61, no. 1, 2011.

[99] W.-B. Pöttner, H. Seidel, J. Brown, U. Roedig, and L. Wolf, "Constructing schedules for time-critical data delivery in wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 10, no. 3, 2014.

[100] M. Doddavenkatappa, M. C. Chan, and B. Leong, "Splash: Fast data dissemination with constructive interference in wireless sensor networks," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2013.

[101] J. P. Martin and L. Alvisi, "Fast byzantine consensus," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, 2006.

[102] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the USENIX Annual Technical Conference (ATC)*. USENIX Association, 2014.

[103] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: An engineering perspective," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2007.

[104] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, and C. Frost et al., "Spanner: Google's globally-distributed database," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2012.

[105] M. Isard, "Autopilot: Automatic data center management," Microsoft, Tech. Rep., 2007.

[106] J. Rao, E. J. Shekita, and S. Tata, "Using Paxos to build a scalable, consistent, and highly available datastore," *VLDB Endowment*, vol. 4, no. 4, 2011.

[107] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: A short preamble mac protocol for duty-cycled wireless sensor networks," in *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. ACM, 2006.

[108] P. J. Marandi, M. Primi, N. Schiper, and F. Pedone, "Ring paxos: A high-throughput atomic broadcast protocol," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2010.

[109] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems (3rd Ed.): Concepts and Design*. Addison-Wesley Longman Publ., 2001.

[110] C. Adjih, E. Baccelli, E. Fleury, and G. Harter et al, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Proceedings of the IEEE World Forum on Internet of Things (WF-IoT)*. IEEE, 2015.

[111] Texas Instruments, "Chipcon CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver," 2006.

[112] R. Lim, F. Ferrari, M. Zimmerling *et al.*, "Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Proceedings of the IEEE/ACM International Conference on Information Processing in Sensor Networks (IPSN)*. ACM, 2013, pp. 153–166.

[113] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1999.

[114] L. Lamport, "Fast Paxos," *Distributed Computing*, vol. 19, no. 2, 2006.

[115] L. Lamport and M. Massa, "Cheap Paxos," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2004.

[116] Q. Wang, X. Vilajosana, and T. Watteyne, "6TiSCH Operation Sublayer Protocol (6P)," IETF, IETF draft-ietf-6tisch-6top-protocol-12, 2018.

[117] M. Van Middlesworth, K. Dresner, and P. Stone, "Replacing the stop sign: Unmanaged intersection control for autonomous vehicles," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 3, 2008, pp. 1381–1384.

[118] F. Dressler, C. Sommer, D. Eckhoff, and O. K. Tonguz, "Toward realistic simulation of intervehicle communication," *IEEE Vehicular Technology Magazine*, vol. 6, no. 3, pp. 43–51, 2011.

[119] M. Segata, B. Bloessl, S. Joerer, C. Sommer, M. Gerla, R. Lo Cigno, and F. Dressler, "Toward communication strategies for platooning: Simulative and experimental evaluation," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 12, pp. 5411–5423, 2015.

[120] H. Cao, S. Gangakhedkar, A. R. Ali, M. Gharba, and J. Eichinger, "A 5G V2X testbed for cooperative automated driving," in *Proceedings of the IEEE Vehicular Networking Conference, (VNC)*. IEEE, 2017.

[121] I. U. Rasool, Y. B. Zikria, and S. W. Kim, "A review of wireless access vehicular environment multichannel operational medium access control protocols: Quality-of-service analysis and other related issues," *International Journal of Distributed Sensor Networks*, vol. 13, 2017.

[122] S. Adams and M. J. Rutherford, "Decentralized intersection management through peer-to-peer technology," *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P)*, pp. 1–5, 2015.

[123] C. Sommer, F. Hagenauer, and F. Dressler, "A networking perspective on self-organizing intersection management," in *Proceedings of the IEEE World Forum on Internet of Things (WF-IoT)*. IEEE, 2014, pp. 230–234.

[124] M. Ferreira and P. M. D'Orey, "On the impact of virtual traffic lights on carbon emissions mitigation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 284–295, 2012.

[125] F. Hagenauer, P. Baldemaier, F. Dressler, and C. Sommer, "Advanced Leader Election for Virtual Traffic Lights," *ZTE Communications, Special Issue on VANET*, vol. 12, no. 1, pp. 11–16, 2014.

[126] A. A. Hassan and H. A. Rakha, "A Fully-Distributed Heuristic Algorithm for Control of Autonomous Vehicle Movements at Isolated Intersections," *International Journal of Transportation Science and Technology*, vol. 3, no. 4, pp. 297–309, 2014.

[127] R. Naumann, R. Rasche, J. Tacken, and C. Tahedl, "Validation and simulation of a decentralized intersection collision avoidance algorithm," in *Proceedings of the IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 1997, pp. 818–823.

[128] M. Choi, A. Rubenecia, and H. H. Choi, "Reservation-based cooperative traffic management at an intersection of multi-lane roads," in *Proceedings of the International Conference on Information Networking (ICOIN)*, 2018, pp. 456–460.

[129] "MSP430 Ultra-Low-Power MCUs." [Online]. Available: http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html

[130] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2004.

[131] R. Jacob, J. Baechli, R. D. Forno *et al.*, "Synchronous transmissions made easy: Design your network stack with baloo," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2019, pp. 106–117.

[132] D. Carlson, M. Chang, A. Terzis *et al.*, "Forwarder selection in multi-transmitter networks," in *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, May 2013, pp. 1–10.

[133] C. Sarkar, R. V. Prasad, R. T. Rajan *et al.*, "Sleeping beauty: Efficient communication for node scheduling," in *IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, Oct 2016, pp. 56–64.

[134] X. Ma, P. Zhang, Y. Liu *et al.*, "Competition: Using DeCoT+ to collect data under interference," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2019, pp. 290–291.

[135] C.-J. M. Liang, N. B. Priyantha, J. Liu *et al.*, "Surviving wi-fi interference in low power zigbee networks," in *Proceedings of the ACM conference on embedded networked sensor systems (SenSys)*. ACM, 2010, pp. 309–322.

[136] Kiam Heong Ang, G. Chong, and Yun Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.

[137] S. Ryu and C. M. Rump, "Application of a pid feedback control algorithm for adaptive queue management to support tcp congestion control," *Journal of Communications and Networks*, vol. 6, no. 2, pp. 133–146, 2004.

[138] C.-K. Chen, H.-H. Kuo, J.-J. Yan, and T.-L. Liao, "Ga-based pid active queue management control design for a class of tcp communication networks," *Expert Systems with Applications*, vol. 36, no. 2, pp. 1903–1913, 2009.

[139] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the American Mathematical Society*, vol. 58, no. 5, pp. 527–535, 1952.

[140] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Proceedings of IEEE Annual Foundations of Computer Science*, 1995, pp. 322–331.

[141] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM journal on computing*, vol. 32, no. 1, pp. 48–77, 2002.

[142] T. Istomin, C. Kiraly, and G. P. Picco, "Is RPL ready for actuation? a comparative evaluation in a smart city scenario," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. Springer, 2015, pp. 291–299.

[143] F. Mager, R. Jacob, R. D. Forno *et al.*, "Competition: Low-power wireless bus baseline," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2019, pp. 292–293.

[144] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proceedings of theInternational Conference on International Conference on Machine Learning (ICML)*, 2016, pp. 2849–2858.

[145] C. A. Boano, T. Voigt, C. Noda *et al.*, "JamLab: Augmenting sensornet testbeds with realistic and controlled interference generation," in *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, Apr. 2011, pp. 175–186.

[146] M. Schuß, C. A. Boano, M. Weber *et al.*, "A competition to push the dependability of low-power wireless protocols to the edge," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, Feb. 2017, pp. 54–65.

[147] A. Forster and A. L. Murphy, "FROMS: Feedback routing for optimizing multiple sinks in wsn with reinforcement learning," in *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information (ISSNIP)*, Dec 2007, pp. 371–376.

[148] E. Ancillotti, C. Vallati, R. Bruno *et al.*, "A reinforcement learning-based link quality estimation strategy for RPL and its impact on topology management," *Computer Communications*, vol. 112, no. C, pp. 1–13, Nov. 2017.

[149] J. Zhu, Y. Song, D. Jiang, and H. Song, "A new deep-Q-learning-based transmission scheduling mechanism for the cognitive internet of things," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2375–2385, Aug 2018.